Misalignment Tolerant Inductive Power Transfer (IPT) Systems

System analysis and controller design

Authors:
D. Ha
M.J.F. Pieters
Delft University of Technology

as part of the BSc graduation project

Dr. Jianning Dong, Supervisor
Soumya Bandyopadhyay (PhD Candidate), Supervisor

Delft, TU Delft, 2017

Additional files are available from mikepieters.com/IPT.zip

Acknowledgements

This report has been written in the context of the Bachelor Graduation Project from the bachelor Electrical Engineering. It describes the outcome of the past quarter in detail. This report would not have been possible without the guidance, support and help of our supervisors Dr. Jianning Dong and Soumya Bandyopadhyay. We would like to express our sincere gratitude towards them for giving us the opportunity to take part in a very interesting and cutting-edge research project, their open attitude towards the team, the constructive feedback on our work, and the generosity in sharing their time.
Last but not least, we would like to thank dr.ir. Nick van der Meijs and Ir. J.M.A. Kooijman, for giving us the opportunity and the trust to start working on this project. Without them, this report would not even have been possible in the first place.

Damy and Mike June 2017

Executive Summary

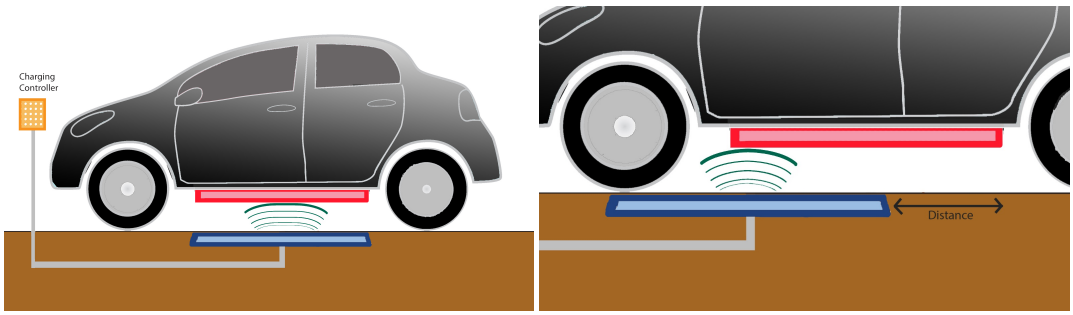
This report contains the design and implementation of two control systems related to an inductive power transfer system. This IPT system charges a battery at maximum efficiency and constant power using a buck converter. A linear controller has been designed around the buck converter and implemented on a FPGA to match impedance for maximum efficiency. A second controller has been designed at the inverter to implement charging at constant power. The control signals for the inverter at the primary side and the PWM generator for the buck converter are also implemented on a FPGA and tested. The control systems have only been simulated.

Contents

1 Problem Definition

1.1 Problem Scope

Electric vehicles are currently charged at EV charging stations through cable connections. New charging applications for these vehicles are required to make charging easier. Inductive Power Transfer (IPT) is gaining popularity among the public. IPT has been used to charge various electronic devices - i.e. phones, watches and toothbrushes. IPT can also be used to charge electric vehicles batteries. In figure 1.1 a simplified inductive power transfer system is shown for an electric vehicle. A coil is installed underneath the car (red square). By applying an alternating voltage on the primary coil (blue coil) and by bringing the secondary coil close to the primary coil, a current is induced in the secondary coil. This current charges the battery inside the electric vehicle. Charging electric vehicles comes with its own kind of problems. One of these difficulties occurs when the coil underneath the car is misaligned with respect to the primary coil. Figure 1.1(b) shows misalignment into one direction, but misalignment can occur in both x and y plane. Misalignment causes more losses in the system and thus affects efficiency. To compensate for the effects due to misalignment, a system should be implemented that takes care of the effects. Designing this compensation part of the inductive power transfer system will be the main goal of this report.



(a) The car when perfect aligned (b) The car when misaligned

Figure 1.1: Misalignment of a car in an IPT system. [1]

1.2 Technical review

Current car charging technology requires the user to plug a charging cable in the car. Implementations of charging the car without cable are currently being investigated. If these wireless charging system were to replace the conventional charging systems, it would require that the wireless system to be as capable as conventional systems. High-end car chargers are currently able to charge batteries at 22kWh with 100A [2].

In conventional battery charing literature there are two main types of charging a battery which are slow charge and fast charge [3]. Slow charge is used when devices are left idle for a long time. The devices charges slowly at the benefit that the battery is not damaged when the battery has reached maximum charge. Fast charge is applied when a device must charge within a certain time limit. The drawback of fast charging is that the battery is potentially damaged if its fast charging for a long time. On average an efficiency is achieved around 80% to 90% [3].

In this report neither fast nor slow charge will stand central but efficiency. In IPT systems, high efficiencies can only be achieved if hardware is present that deals with efficiency. The efficiency can greatly depend on the misalignment of the coils.

1.3 Design Requirements

The source of the design requirements are set by our supervisor S. Bandyopadhyay. The following design requirements must be met:

1. A control system must be designed that achieves maximum efficiency: $\eta = P_{battery}/P_{source}$.

2. The control system should achieve maximum efficiency for misalignments in the range of k = $0.08 \leq k \leq 0.20$, where k is the coupling coefficient between the coils.

3. The battery must be charged with 8.0kW during steady state regardless of the misalignment.

4. An additional task would be to implement the controller on a FPGA.

5. An additional task would be to implement a second controller that can set the power at which the battery is charging.

Parameters of the conventional IPT system that has already been built is given in table 1.1.

Variable	Value	Description
$V_{battery}$	200V	The nominal battery voltage
L_1	200μH	Inductance at the primary side of the transformer
L_2	200μH	Inductance at the secondary side of the transformer
C_p	$\frac{1}{\omega^2 L_1}$F	Capacitance at the primary side of the transformer
C_s	$\frac{1}{\omega^2 L_2}$F	Capacitance at the secondary side of the transformer
ω	2π85k rad/s	Angular frequency of the primary voltage source
$R_s = R_p$	-	The parallel resistances in the primary and secondary coil are the same
k	$0.08 \leq k \leq 0.20$	The range of the coupling coefficient
P_{out}	8kW	The power output of the IPT system

Table 1.1: The parameters of the IPT system

2 Design Description

2.1 Overview

Figure 2.1 shows the overview of the IPT system. The IPT system consists of a source that transmits energy from a primary coil to a secondary coil (transformer). Reactive compensation is implemented to bring voltage and current in phase. The current received at the secondary coil is rectified and a capacitor creates a DC voltage. In order to improve efficiency $\eta = \frac{P_{bat}}{P_{source}}$ at different misalignments a DC/DC converter is introduced.

The DC/DC converter has the possibility to change the ratio between voltage $U_{2,DC}$ and current $I_{2,DC}$ via the duty cycle and thus match impedance. How this exactly works is explained in section 2.2 and is verified in section 2.3. In order to get the right duty cycle, a control system must be implemented. The design control system is given in section 2.5 to section 2.10. Work group 1 will focus on the design of the DC/DC converter. Work group 2 will focus on supplying the measured signals to the control work group. Work group 3 will design the control system that matches impedance. An additional controller can also be designed that controls the voltage source at the primary side of the coil. Controlling the voltage source results in the ability to set the power at which the battery is charging.

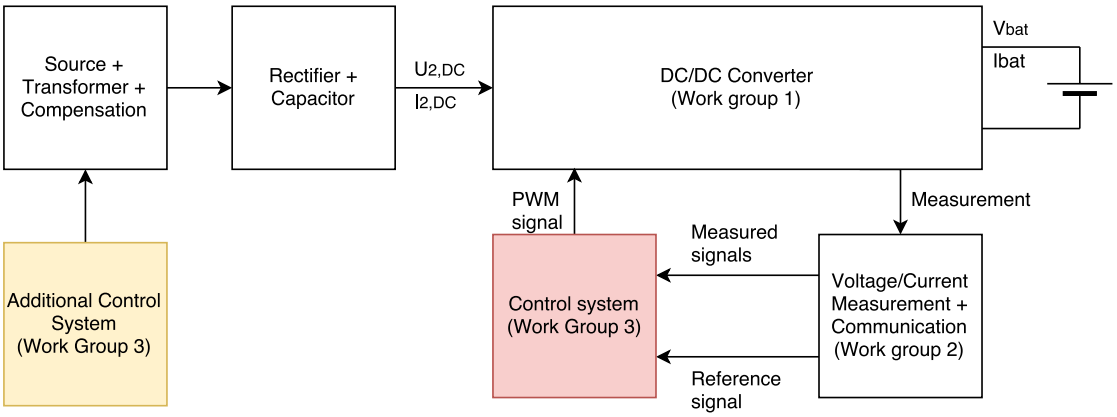


Figure 2.1: Overview of connection between sub groups. This document focuses on the tasks of the red square. The yellow square is an additional controller which could be designed.

2.2 Problem analysis

A general IPT system can be modeled as figure 2.2. V_p is the primary AC voltage source. Capacitors C_p and C_s improve the power factor. Capacitors C_p and C_s are chosen such that the voltage and current at the rectifier are in phase. The transformer is coupled with factor k which changes depending on the misalignment. M is the mutual inductance, which is given by $M = kL$, where k is the coupling coefficient and L the inductance of the coils. The coils have the same inductance. The rectifier creates a DC voltage from its input, such that battery can charge at a constant voltage. The coils and capacitors have a parasitic resistance which are summed up as R_p for the primary side and R_s on the secondary side. These parasitic resistances affect the efficiency $\eta = \frac{P_{bat}}{P_{source}}$.

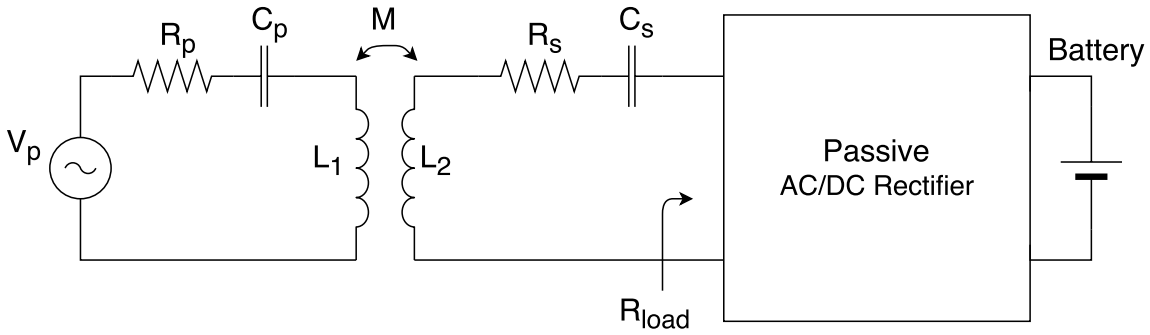


Figure 2.2: The equivalent circuit of an IPT system with DC output.

The proposed method to get maximum efficiency is to use a DC to DC converter to match impedances. The DC to DC converter, which will be placed between the rectifier and battery in figure 2.2, causes the impedance R_{load} to change. This results in a circuit with an equivalent

resistance R_{load} as can be seen in figure 2.3.

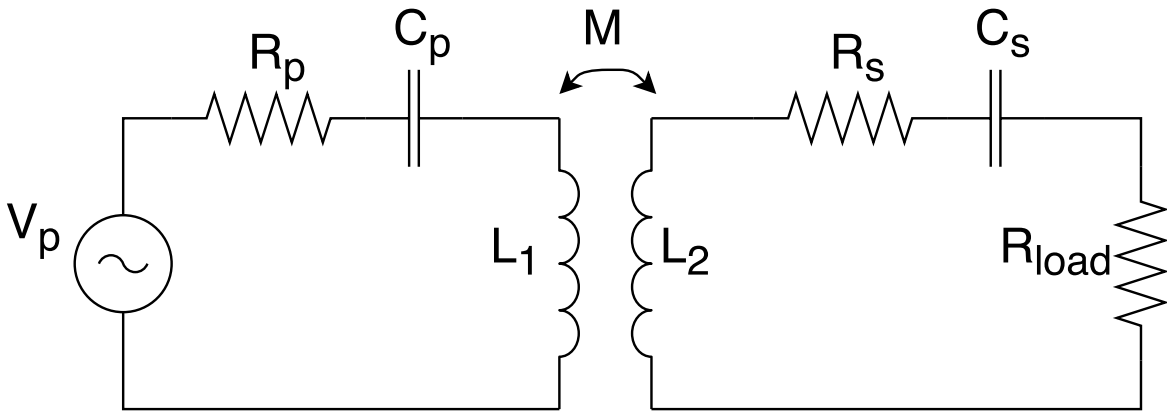


Figure 2.3: IPT system with equivalent resistance R_{load}.

Calculations (Appendix A) show that the efficiency η is given by equation 2.1, given that $L_1 = L_2 = L$ and $C_p = C_s = \frac{1}{\omega^2 L}$ (table 1.1). Setting the derivative of equation 2.1 with respect to R_{load} to zero yields the load with optimal efficiency which is given by equation 2.2. Since the primary coil has the same properties as the secondary coil; $R_s = R_p$. The calculations can be found in appendix A. Maximum efficiency is thus achieved when R_{load} is equal to ωM for any misalignment.

$$\eta = \frac{P_{load}}{P_{source}} = \frac{\omega^2 M^2 R_L}{(R_s + R_{load})^2 R_p + \omega^2 M^2 (R_s + R_{load})} \tag{2.1}$$

$$R_{load,opt} = \omega M \sqrt{\frac{R_s}{R_p}} = \omega M \tag{2.2}$$

2.3 Verification of the maximum efficiency point

Figure 2.3 is simulated in Simulink using values from table 1.1. A sweep is performed on R_{load} with steps of 1Ω at different values for k. The Matlab code is available in appendix B. The result of this simulation is depicted in figure 2.4. The dots is where the maximum efficiency is expected. The graph shows the expected outcome.

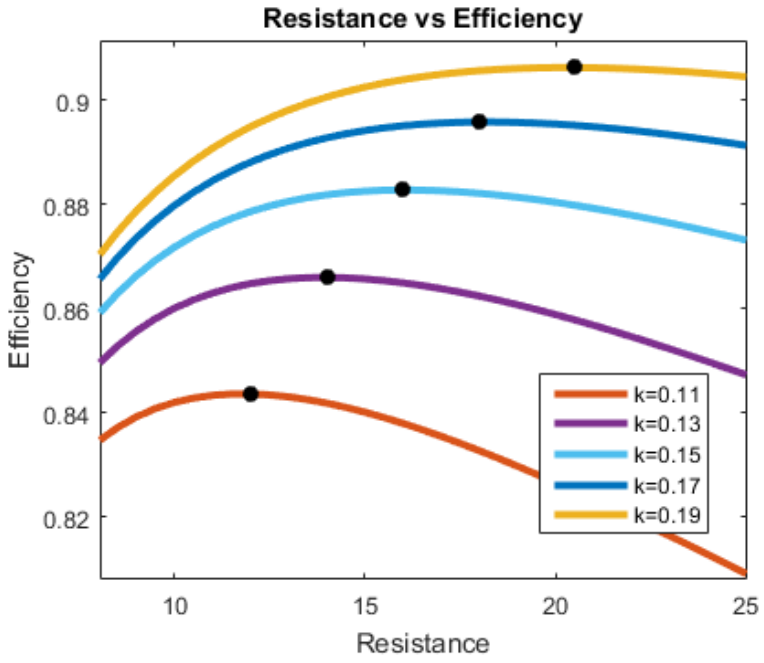


Figure 2.4: The efficiency versus resistance plot produces by the code in appendix B

2.4 Comparison of converters

Assume we want to charge a load with a certain power $P_{out} = I_{battery}V_{battery}$. To change the ratio between the current and the voltage a DC/DC converter can be used. Using a DC/DC converter will allow changing the input voltage $U_{2,DC}$ as in figure 2.5. Assuming the converters will all be in continuous conduction mode, the following characteristics of the converters can be derived [4].

Buck converter in CCM

A buck converter takes a relatively high input voltage and reduces it at the output:

$$V_{out} = DV_{in}, \quad I_{out} = \frac{1}{D}I_{in} \tag{2.3}$$

Since $0 \leq D \leq 1$, the following constraint can be derived:

$$V_{in} \geq V_{out} \tag{2.4}$$

Boost converter in CCM

A boost converter takes a relatively low input voltage and amplifies it:

$$V_{out} = \frac{V_{in}}{1-D}, \quad I_{out} = (1-D)\, I_{in} \tag{2.5}$$

Where $0 \leq D \leq 1$ Now the in and output resistances can be written as: Since $0 \leq D \leq 1$, the following constraint can be derived:

$$V_{in} \leq V_{out} \tag{2.6}$$

Note that both equations are valid for buck and/or boost converters with zero losses. Practically, these converters don't exists. This is why the boundaries for the voltages are too loose, but they can be used as a guideline. In the following sections, equations for the input voltage of the buck - $U_{2,dc}$ - will be derived, see eq. 2.12. From this formula and table 1.1. The following boundaries for the input voltage can be obtained (assuming $\eta_{rect} = 1$ and $M = L_p k = L_s k$):

$$U_{2,dc,max} = \sqrt{\frac{\pi^2}{8} \omega L_p \max(k) P^*_{out}} \approx 459.18V \tag{2.7}$$

$$U_{2,dc,min} = \sqrt{\frac{\pi^2}{8} \omega L_p \min(k) P^*_{out}} \approx 290.41V \tag{2.8}$$

This shows $U^*_{2,dc}$ will be higher than the nominal battery voltage U_{batt}. However, the internal resistance over the battery raises the voltage while charging. Since the internal resistance of the battery will be relatively low, the buck converter topology is chosen because of its voltage reducing characteristics, $U_{2,dc} \geq V_{batt}$.

2.5 The control topology

The purpose of this section is to explain the control topology that has been chosen.

As has been stated before, the system contains two controllers. One controller implements maximum efficiency by controlling the buck converter. The other controller is used to charge the battery at a constant power. The derivation of the control topology for both controllers is explained in this section. This section starts with the derivation for the efficiency controller and then derives the topology of the power controller.

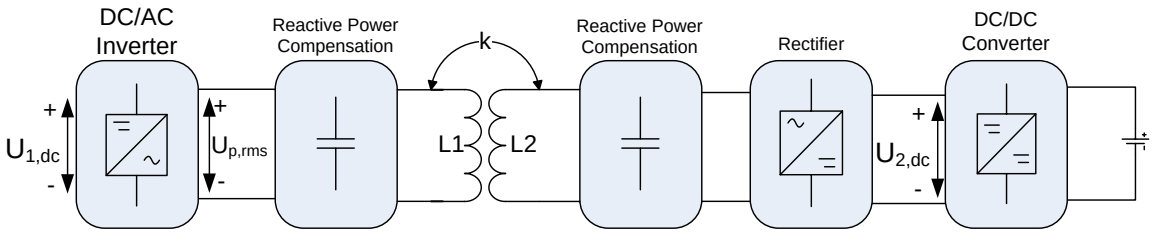


Figure 2.5: Systematic overview of the IPT system. From inverter to battery.

Take R_{load} at the same position as in figure 2.3, but now in figure 2.5. Impedance matching occurs when $R_{load} = \omega M$. R_{load} can be related to $U_{2,dc}$ (the voltage after the rectifier), the rectifier efficiency η and the power out P_{out}. This is done by modeling the resistance before and after the rectifier as $R_{ac} = R_{load}$ and R_{buckin} respectively [5, 6]. This is made visible in figure 2.6. For now the DC/DC converter is modeled with no losses. The relation between R_{ac} and R_{buckin} is given by equation 2.9 [5, 6].

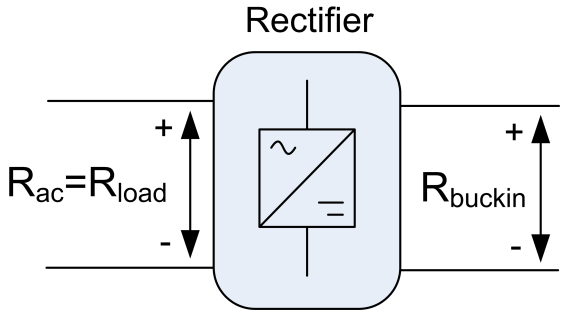


Figure 2.6: Rectifier with input and output resistances

$$R_{ac} = \frac{8}{\pi^2 \eta_{rect}} R_{buckin} \tag{2.9}$$

R_{buckin} can be written as equation 2.10.

$$R_{buckin} = \frac{U_{2,dc}}{I_{2,dc}} = \frac{U_{2,dc}^2}{P_{out}} \tag{2.10}$$

Combining equations 2.9 and 2.10 yields equation 2.11.

$$R_{ac} = \frac{8}{\pi^2 \eta_{rect}} \frac{U_{2,dc}^2}{P_{out}} \tag{2.11}$$

The design specifications state that the battery should charge with a fixed power. This means that there will be a power set-point P^*_{out}. This is the power at which the battery has to be charged. The battery charges at maximum efficiency ($R_{ac} = \omega M$) which leaves $U_{2,dc}$ as the only variable left. Solving $U^*_{2,dc}$ yields the voltage set-point which is given by equation 2.12. With this set-point it can be concluded that the voltage over the input capacitor must be controlled for maximum efficiency. Design requirement 1 and 2 can now be fullfiled.

Theoretically it could also be possible to control the current $I_{2,DC}$. Literature shows that in this voltage-mode control has better properties due to the power at which the battery must charge [7].

$$U^*_{2,dc} = \sqrt{\frac{\pi^2 \eta_{rect}}{8} \omega M P^*_{out}} \tag{2.12}$$

Controlling $U^*_{2,dc}$ causes the system to charge at maximum efficiency assuming that the voltage delivered by the inverter is correct. In order to charge both charge the battery at the desired power and to reach the maximum efficiency point, the voltage at the output of the DC/AC inverter - $U_{p,rms}$ as shown in figure 2.5 - should also be controlled.

To control $U_{p,rms}$ the modulation scheme proposed in [8] is used. [8] describes a full bridge inverter and modifies the width of the positive and negative pulses of the signal, such that the RMS voltage at the output of the inverter is lower than the input. This modulation scheme is depicted in figure 2.7, where α/ω is the width of the positive and negative signal after the inverter.

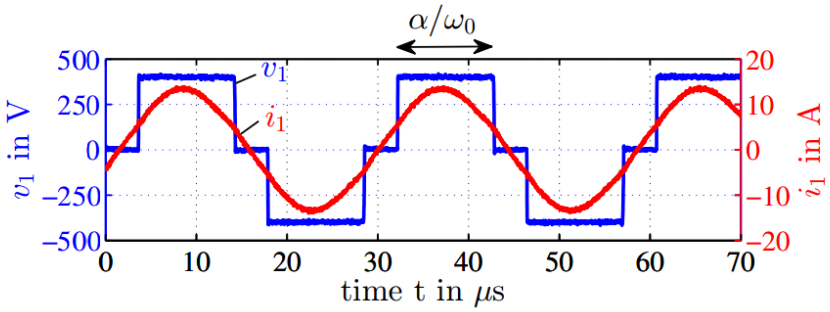


Figure 2.7: Modulation to control $U_{p,rms}$ [8].

Power transfer of the complete IPT system can be written as equation 2.13 [8]:

$$P_{out} = \frac{8}{\pi^2} \frac{U_{1,dc} U_{2,dc}}{\omega M} \sin\frac{\alpha}{2} \tag{2.13}$$

Rewriting equation 2.13 to α yields the set-point α^* at maximum efficiency and at the desired power (when $U_{2,dc}$ is also U^*_{dc}). This α^* is given by by 2.14 [8]. Controlling α to α^* thus yields the desired power. Design requirement 3 and 5 can now be be fullfilled.

$$\alpha^* = 2\sin^{-1}\left(\frac{\pi^2}{8}\frac{\omega M P^*_2}{U_{1,dc}U^*_{2,dc}}\right) \tag{2.14}$$

$$U^*_{1,dc} = \sqrt{\frac{L_p}{L_s}}U^*_{2,dc} \tag{2.15}$$

Since the range of k is known (table 1.1) $U_{1,dc}$ can also be calculated. Using equation 2.15 from [9], $U_{1,dc}$ should be the maximum value of what $U_{2,dc}$ can be. This is because the output of the inverter can only be made lower by changing α, but not any higher. Given that $L_p = L_s$ yields equation 2.16.

$$U_{1,dc} = \max(U_{2,dc}) = \sqrt{\frac{\pi^2\eta_{rect}}{8}\omega\max(M)P^*_{out}} = \sqrt{\frac{\pi^2\eta_{rect}}{8}\omega L_{\text{transformer}}\max(k)P^*_{out}} \approx 460 \tag{2.16}$$

460V holds for an inverter with no losses and $\eta_{rect} = 1$. This is the minimum value for $U_{1,dc}$ in order to work. The actual voltage value should be chosen bit higher to account for the losses in the circuit.

The only issue that remains is that k has been taken in the range of 0.08 and 0.20 (table 1.1. The assumption is made that this value of k is supplied. The real system needs an extra subsystem that calculates this coupling coefficient real time. The real-time calculation of k is outside of the scope of this work group.

The equations and concepts can now be used to create the overview for the complete system, together with the controller and set point calculations. This yields figure 2.8.

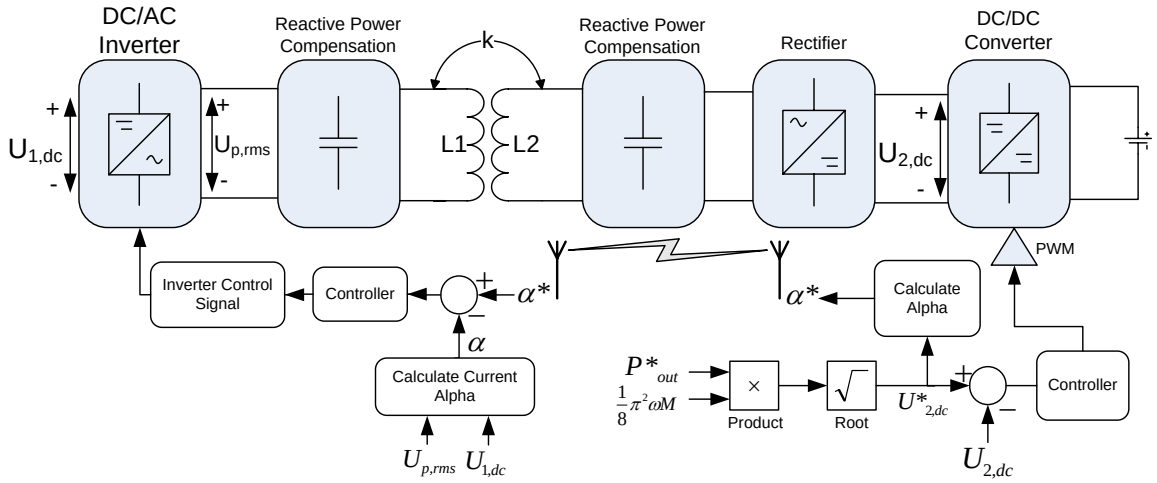


Figure 2.8: Systematic overview of the IPT system. From inverter to battery with both control systems.

2.6 Analysis of the Buck converter

2.6.1 Simplified buck model

The input source of the buck converter can be modeled as a current source due to the transformer. The transformer acts like a current source. The value of this current source at input of the buck converter 'I_s', will be left open for now. For simplification purposes the battery is modeled as a DC source in series with a resistor [10]. This yields the circuit in figure 2.9.

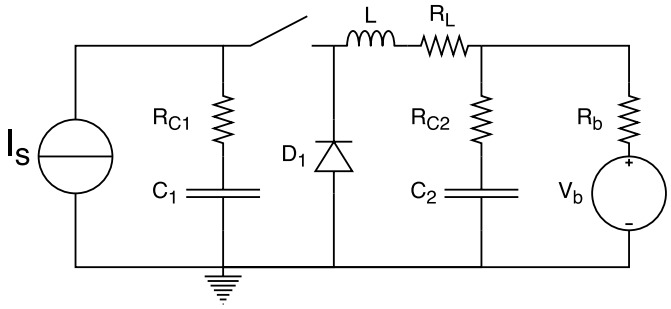


Figure 2.9: Simplification of the buck converter. R_L, R_{C1} and R_{C2} are parasitic resistances. Consider the switch and diode ideal.

The circuit also contains parasitic resistances of the capacitors and inductor. The parasitic resistances add an additional zero to the circuit which interferes with the rise-time. The switch and diode are considered ideal to simplify the problem, as has been done in [11]. The current

through the diode when the switch is closed is negligible.

2.6.2 Average state space model

Literature suggests the use of an average state space model [11, 12]. This state space model allows the steady state analysis of the buck converter. The average state space model can be found by taking the average of the state space equations when the switch is open or closed. This is displayed in equation 2.17. Figure 2.10 shows the circuit when the switch is open or closed.

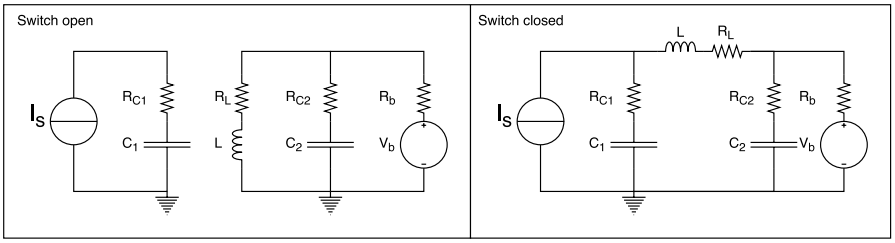


Figure 2.10: Buck converter when switch is open or closed.

The calculations of the state space models when the switch is open and closed can be found in appendix A.2. The resulting average state space equation is given in equation 2.18 making use of [13].

$$\mathbf{X_{avg}}' = \mathbf{X}_{on}d + \mathbf{X}_{off}(1-d) = \mathbf{A}_{avg}\boldsymbol{X} + \boldsymbol{B}_{avg} \tag{2.17}$$

$$\begin{bmatrix} V_{C1} \\ I_L \\ V_{C2} \end{bmatrix}' = \begin{bmatrix} 0 & \frac{-d}{c_1} & 0 \\ \frac{d}{L} & \frac{-1}{L}(R_L + R_{C1}d + R_{C2} \parallel R_b) & \frac{-R_b}{L(R_{C2}+R_b)} \\ 0 & \frac{R_b}{C_2(R_{C2}+R_b)} & \frac{-1}{C_2(R_{C2}+R_b)} \end{bmatrix} \begin{bmatrix} V_{C1} \\ I_L \\ V_{C2} \end{bmatrix} + \begin{bmatrix} \frac{I_s}{C_1} \\ \frac{1}{L}(I_s R_{C1} d - \frac{R_{C2}V_b}{R_{C2}+R_b}) \\ \frac{V_b}{C_2(R_{C2}+R_b)} \end{bmatrix} \tag{2.18}$$

Notice that entry matrix A_{avg} contains entries related to d. This will cause the poles of any transfer function to be related to d. Also notice that entry (2,1) of B_{avg} is the only entry that contains d, and that this entry is not fully wrapped by d. These dependencies on d causes the system to be a nonlinear time invariant system. Nonlinearity is to be expected, since a small duty cycle will cause more charge to build up in the capacitor compared to a large duty cycle, which will cause less charge to build up in the capacitor.

Controllability is checked via $\mathbf{C} = [BABA^2B]$. Matlab determined that the rank of matrix $\mathbf{C}$ is three independent of the value of d, so the system is controllable.

Section 2.5 explained that the voltage over the rectifying capacitor will be controlled. The

average capacitor voltage, which derivation can be found in appendix A.2, can be written as equation 2.19.

Simulations show that the input current I_s is given by $I_s(t) = \left|I_{max}\sin(2\pi f_s t)\right|$, where f_s is the frequency of the source at the primary side of the transformer. The average state space model assumes an average current as input. I_s will be taken as the time average of $I_s(t)$. This means that model only represents the average current. The actual varying current will not cause large deviation in the result, since the goal of the control system is to do an average impedance matching.

Section 2.5 also explained that another controller is present in the system that handles the constant power. This controller is able to control the current going into the rectifier and thus I_s can be any value to a certain extent. Simulations show that if impedance is to be matched at 8.0kW an input current is needed of between 20A and 35A for a k of 0.20 and 0.08 respectively. For now I_s is kept as an variable.

$$Y = \mathbf{C}_{avg}\mathbf{X} + D_{avg} = \begin{bmatrix} 1 & -R_{C1}d & 0 \end{bmatrix}\mathbf{X} + I_s R_{C1} \tag{2.19}$$

Equation 2.20 shows the Laplace transform of the output signal. As can be seen it has a non linear dependency on d. The system is stable, since all the poles are in the right half plane. The system can be linearized which yields equation 2.21. The system has been linearized on $d = 0.7$, since that's the middle of the working region of the switch : $[0.5 - 0.9]$.

$$Y = \frac{1e-3s^3 + (2.87e3d + 3.93e3I_s)s^2 + (3.83e8d + 1.95e6I_s)s + 2.25e10I_s + 8.74e12d}{s^3 + (3.69e3)s^2 + (1.76e7d^2 + 6.31e7)s + 2.19e10d^2} \tag{2.20}$$

$$Y = \frac{-1.42e-2I_s^2 s^3 - 1.975e5s^2 + (2.705e7Is + 6.793e9)s + 8.741e12}{s^3 + 4.00e3s^2 + 7.22e7s + 1.07e10d^2}d \tag{2.21}$$

The system remains stable. Since the linearized system is a T1-system a PI controller can be used as a first guess to get the steady state error to zero. Note that d now must be between $[-1, 0]$ in order to work. This means that the constants of the controller must be negative if d is in $[0, 1]$.

2.6.3 Model Validation

Figure 2.11 shows the output voltage of the average state space model versus the simulation of the buck converter with an rectified sinusoidal current source. The values of table 2.1 have

Variable	Value	Description
R_{c1}	10mΩ	The parasitic resistance of C_1
R_{c2}	20mΩ	The parasitic resistance of C_2
R_l	30mΩ	The parasitic resistance of R_l
C_1	2.8055mF	Capacitance C_1
C_2	789.47μF	Capacitance C_2
L	20.256μH	Capacitance C_2
f_{sw}	500kHz	Inductance L

Table 2.1: Buck constants supplied by WP-1

been used as constants for the buck converter. I_s has been chosen as 25A.

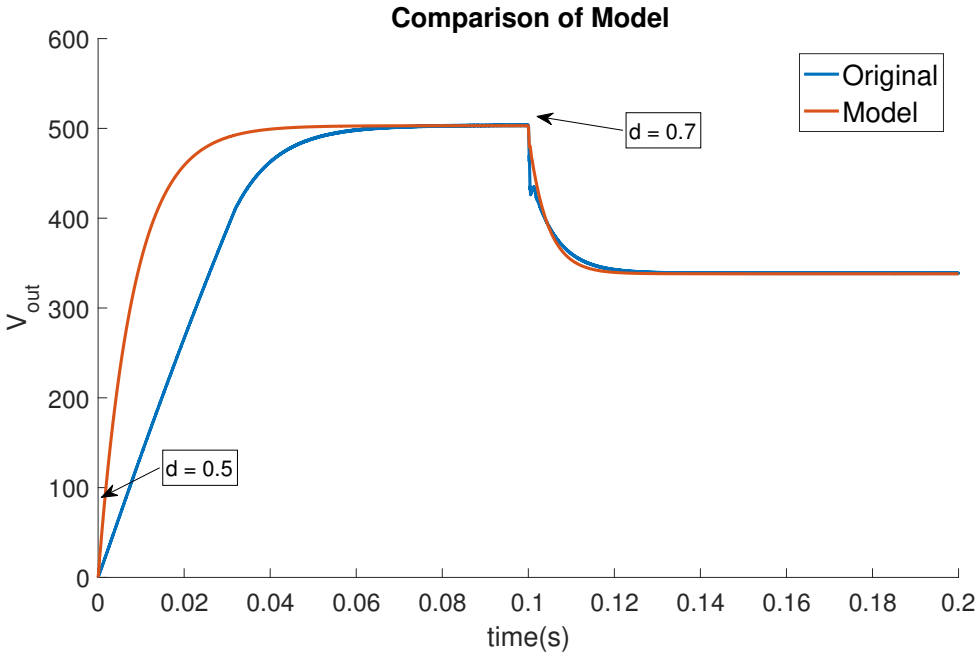


Figure 2.11: Comparison of models with switching duty cycle.

Figure 2.11 shows that the rise time of the model is not representative of the actual rise time. The overshoot of the model however is comparable to the actual system. Since dynamic performance is not the most important task of the controller, the model is can still be used for analysis of steady state performance.

2.7 Control Design

2.7.1 Tasks for the control system

The control system has to implement the following tasks:

- R_{load} must on average be equal to ωM when the coupling factor is between 0.08 and 0.20. This requirement is for a fixed power of 8.0kW.

- Using equation 2.12 the range of which the voltage over the capacitor can be acquired. This range is 260V - 460V for a fixed power (8.0kW).

- The rise time of the system is preferred to be smaller than 500μs for $0.08 \leq k \leq 0.20$.

- The settle time of the system is preferred to be smaller than 1s for $0.08 \leq k \leq 0.20$.

In order to meet the requirements, two controllers have to be implemented in order to both (1) reach the power set-point and (2) reach the maximum efficiency point.

2.7.2 Choosing the secondary controller

Since the linearized buck system is a T1-system, a PI controller is needed at minimum to get the steady state error to zero. The PI-controller must be saturated to prevent duty cycles greater than 1 or smaller than zero.

2.7.3 Anti-windup method

The integrator should stop integrating when the output of the controller is out of the $0 \leq D \leq 1$ range. This is done in order to prevent the integration block from outputting a large integrated error, which will cause a large overshoot in the end. Anti-windup is implemented using a comparator which continuously checks whether the output of the controller is outside of these bounds ($0 \leq D \leq 1$) and stops integration accordingly. This method of stopping the integrator when the output of the controller is larger than a certain bound is referred to in literature as "clamping". [14]

2.8 Tuning constants for the controller

The objective of the following sections is to determine the constants of the controller such that the controller satisfies the design constraints (section 2.7.1). The method used to determine the constants will be explained.

Finding the constants for any PID based controller is a non convex problem. Suppose that the performance of the controller with arbitrary constants is rated with a function called the fitness function and a high rating means a desirable controller. Unlike convex problems, the solution plane will not be a smooth plane going to a single peak, but can be a plane with discontinuities and multiple peaks or valleys. Figure 2.12 illustrates the difference between a convex and non convex problem.

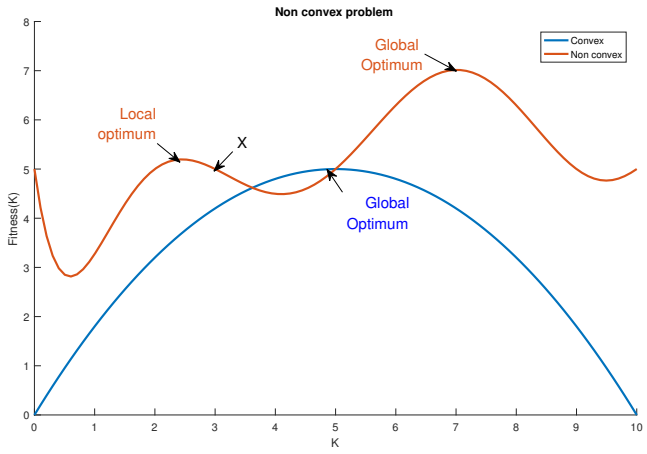


Figure 2.12: Convex vs non convex problem

In a traditional convex problem, gradient descent would be used to move the parameter. The solution will eventually reach the maximum of the fitness function regardless of where the the initial solution starts. In a non convex problem, gradient descent can also be applied. The initial position of the solution however, determines whether the solution will end up in a local optimum or not. Figure 2.12 illustrates this. If the initial solution starts at x, it will climb towards a local optimum. If x would have started at the other side of the valley, it would have reached the global optimum.

Traditional tuning strategies such as Ziegler-Nichols tune PID controllers such that a stable solution is found [15], but not necessarily the solution that is most fit. In order to find this peak in the solution plane, a computational optimization technique has been used. These computational techniques often use randomness (a random value). This randomness is used to move the solution in the solution plane. If the randomness is large, the solution is able to jump over valleys in the function and thus exploring the other side of the valley. This is called exploration. If randomness is small, the solution will be fine tuned. This is called exploitation. The optimization technique that has been used is called: Evolutionary Strategies taken from [16].

2.8.1 Evolutionary Strategies

Evolutionary strategies is a computational optimization technique that is based on evolution. The basic algorithm of this strategy is illustrated in figure 2.13.

In evolutionary strategies the solutions are coded in a vector. For a PI controller, couples of Kp and Ti are chosen according to a strategy. The result would look like $\begin{bmatrix} Kp_i & Ti_i \end{bmatrix}$ and could look something like $\begin{bmatrix} 0.03 & 12.7 \times 10^{-4} \end{bmatrix}$. The set of solutions is called the start population,

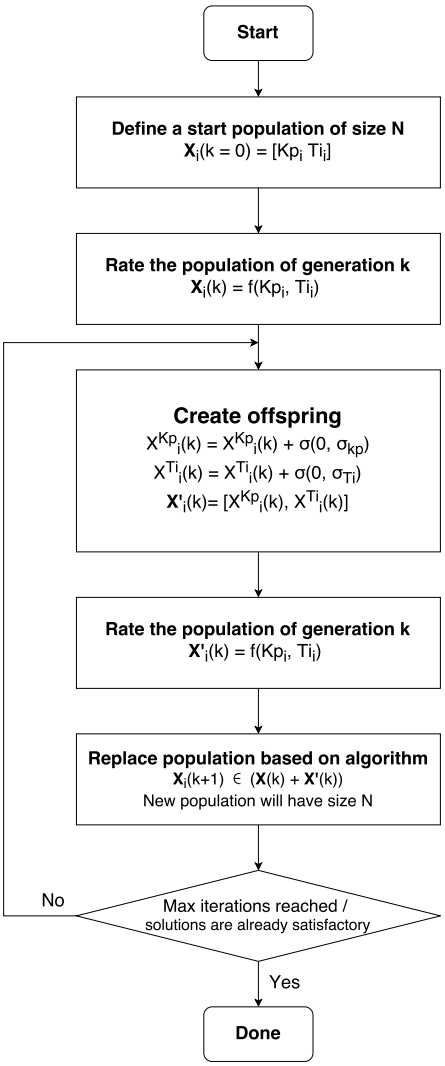


Figure 2.13: Algorithm of evolutionary strategies

i.e. $\begin{bmatrix} Kp_i & Ti_i \end{bmatrix} \in$ StartPopulation. The start population is then rated by putting Kp_i and Ti_i into the controller and assessing the response of the entire system for all i. In Evolutionary strategies the solutions that perform the best are chosen for reproduction. These solutions are then modified, by adding a value from a normal distribution with $\mu = 0$ and $\sigma = C$ to the current solution. These so called offspring are then also rated. Parents and offspring are once more selected for reproduction and then rated. This continues until a certain amount of iterations has passed or until solutions are satisfactory.

A few reasons for choosing evolutionary strategies over other computational strategies are:

- Easy implementation: Evolutionary strategies is relatively easy to implement compared to other strategies. Offspring is generated independent of the other solutions. 'Offspring 'in swarm intelligence requires data of neighboring solutions, which require structures to keep track of that data.
- Small solution space: unlike other computation strategies, evolutionary strategies limits solutions to a certain range, i.e. $Kp_i \in [K_{p,min} K_{p,max}]$, $T_{i,i} \in [T_{i,min} T_{i,max}]$. In subsection 2.8.2 it's argued that it's already known in which range the solution should be.
- Fine tuning randomness: Most other computational optimization techniques only have a constant order of randomness. σ_{K_p} and σ_{T_i} can also be fine tuned. Solutions will now train themselves to exploit or explore the solution space.

2.8.2 Implementation Evolutionary Strategies

Normally the range of the constants K_p and T_i would be from $-$inf to inf. The decision has been made to reduce the search space. If any of the PID constant is too large it will cause the controller to react too fast. This will cause instability if noise is present in the measurement signal. Large constants are thus discarded. It's hard to answer in what range the solution should be.

The output of the controller controls the duty cycle of the PWM generator, which is a value between 0 and 1. If the p term is examined, it's evident that the maximum error that the system could generate is at the start. The maximum error would thus be the highest possible reference voltage. At this point the output of the PI controller should be a value between 0 and 1. As a first guess K_p has been limited to a range of $\left[\frac{-1}{V_{ref,max}}, 0\right]$. This range is slowly increased if K_p turns out to be near the border of range. The same strategy is applied to the I term. The initial range has been chosen as $\left[-1 \times 10^4, 0\right]$. The minimum value has been chosen as such since other papers have similar constants in the -1×10^4 range [17, 18].

The fitness function that has been chosen is given in equation 2.22.

$$f(K_p, T_i) = \min\left(0.5, \frac{1}{2 \times \text{overshoot}_{\%}}\right) + \min\left(0.5, \frac{1}{2 \times e_{ss\%}}\right) \tag{2.22}$$

The fitness function is trying to get the average steady state error to 1% and at the same time the overshoot to 1%. It's important to get the error as close to zero in order to get maximum efficiency. The overshoot is important because excess charge is sent into the battery. This could destroy the battery. The matlab code that implements the genetic algorithm can be found in appendix C.

2.9 Creating control signals for the full bridge inverter

A subsystem has to be created to supply the full bridge inverter with the correct signal to let it create the signal depicted in figure 2.6. α is defined as in figure 2.6. This system should take α and signal frequency f as an input and create two separate control signals for the two transistor pairs in the full bridge inverter. This generator is depicted in figure 2.14.

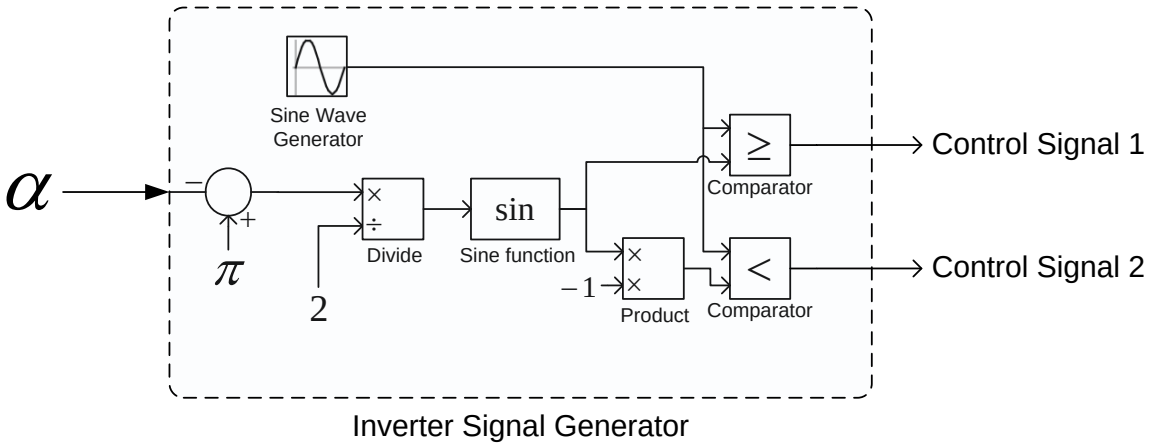


Figure 2.14: Systematic overview of the control signal generator for the full bridge inverter

The working principle behind this generator will now be illustrated by an example. Assume $\alpha = \frac{1}{4}\pi$. This means the nonzero values of the output signal should have a width of $\frac{\pi}{4\omega}$ (figure 2.6). Since our reference signal is a sine-wave, the corresponding reference for both the comparators (x and y) is given by eq. 2.23. Division by 2 is done because the time from 0 to the first high in the control signal, should be half of the total zero time between 0 and π. See figure 2.15.

$$x(\alpha) = \sin\left(\frac{(\pi - \alpha)}{2}\right) = -y(\alpha) \tag{2.23}$$

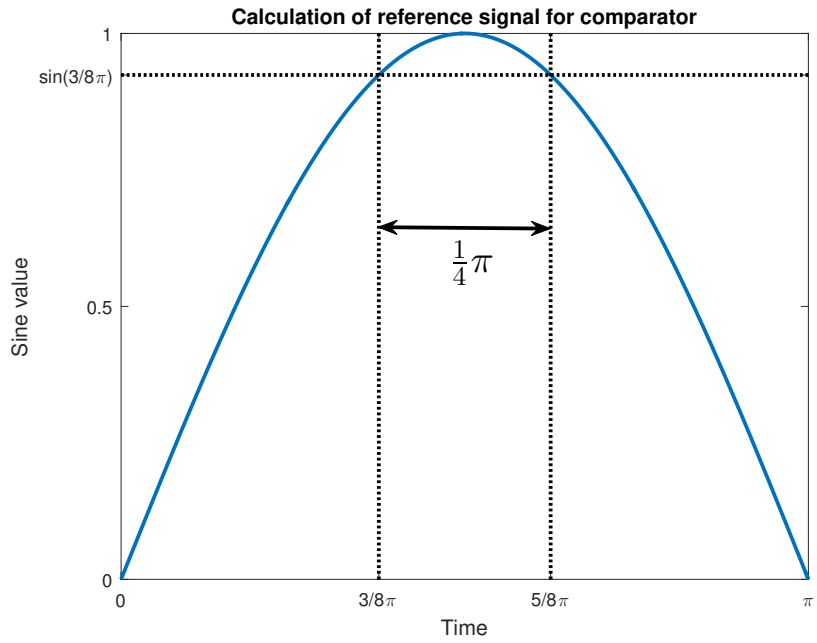


Figure 2.15: Calculation of the reference value for the comparator, when $\alpha = \frac{1}{4}\pi$

2.10 FPGA Implementation

A PI controller and PWM generator have been implemented on a FPGA in order to meet design requirement 4. The FPGA has to be fast - have a high clock frequency - in order to generate an PWM signal with a large enough resolution such that the quantization error is low. The number of quantization levels n is given by equation 2.24.

$$n = \frac{f_{clk}}{f_{PWM}} \tag{2.24}$$

The Papilio One with on board Xilinx Spartan3E (XC3S500E) has been used due to the following [19, 20]: The FPGA is large enough to accommodate the implementation and also has an USB chip for JTAG programming. The FPGA also has an on-board flash memory, which allows the system to work even after a restart. The FPGA is quite affordable. The FPGA also allows the usage of the Xilinx ISE Design suite, which is a all-in-one package for creating and simulation of hardware implementations. [21] has been used as a guide during the design of the PWM generator, the inverter signal generator and the PI-controller.

2.10.1 PWM Generator

The Papilio One development board has an internal clock of 50MHz [20]. The Buck-design group needs an switching frequency of f_{PWM} = 500kHz. Applying equation 2.24 yields a resolution of 64 discrete steps for the duty cycle (D). This means that the PWM generator needs a bit signal of size $^2\log(64) = 6$ in order to represent all 64 steps. The maximum duty cycle is defined by us as "111111" and the minimum "000000". The conversion of duty cycle to binary signal is given by equation 2.25. Figure 2.16 illustrates an example PWM signal.

$$D_{bin} = \text{to\_binary}(D_{\%} \times 64) \tag{2.25}$$

In VHDL the following code has been writen:

```
1 time_high <= conv_integer(dutycycle_IN);
```

Time_high is the number of clockcycles the PWM-signal has to stay a logic high. A counter counts the clock cycles that have passed. After Time_high amount of clock cycles have passed, the PWM-signal is changed to a logic low. Once the counter reaches the number of the total clock cycles per PWM, which is 64, the counter is reset and the PWM-signal is set to a logic high.

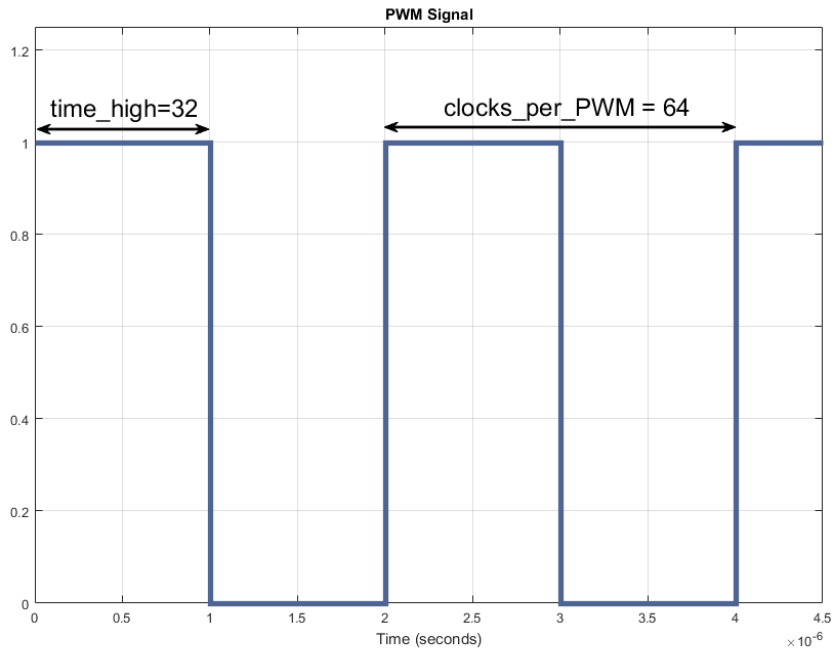


Figure 2.16: PWM signal for a dutycycle of 50%

Synthesizing the code from appendix D.1 creates the model in figure 2.17. The PWM generator is self-explanatory except for one thing. This PWM generator duty cycle can only be changed if and only if the latch is a logic 1. The dutycycle_IN input is read and a new duty cycle is loaded in. This is to keep the duty cycle stable in case the controller is suddenly changing the duty cycle. Testing and verification of the model is done in section 3.4.

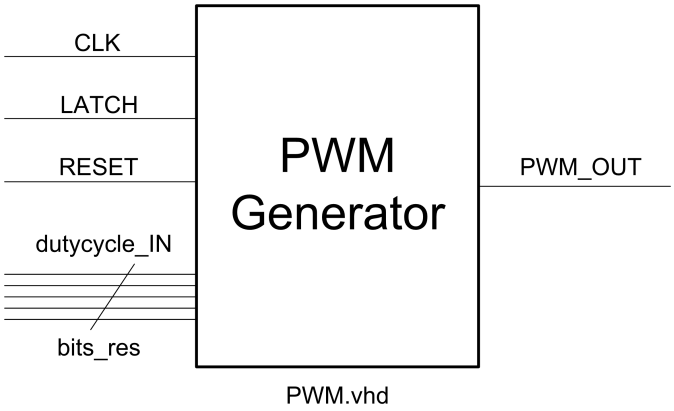


Figure 2.17: Model for PWM generator entity

2.10.2 Control signal generator for inverter

The design for this signal generator is for a large part based on the PWM generator in subsection 2.10.1 with three differences. First, the frequency of the generated signals is f = 85kHz (table 1.1) instead of f_{sw} = 500kHz (table 2.1). Secondly, the maximum equivalent duty-cycle allowed is 50%. Lastly, there should be two output signals that are 180° out of phase.

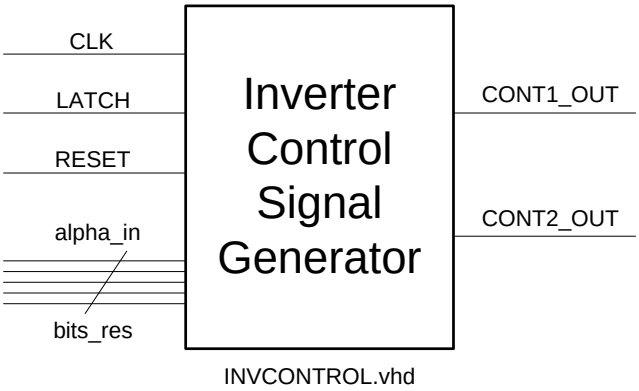


Figure 2.18: Model for inverter controller entity

alpha_IN is calculated in equation 2.26:

$$\texttt{alpha\_IN} = \text{to\_binary}\left(\frac{\alpha}{\pi} * \texttt{clocks\_per\_pulse}\right) \tag{2.26}$$

Where clocks_per_pulse is the number of clock-cycles for maximum alpha. This occurs when $\alpha = \pi$. Which is calculated by 2.27 (pseudocode).

$$\texttt{clocks\_per\_pulse} = \texttt{to\_integer}\left(\frac{\texttt{clk\_freq}}{2 * \texttt{control\_freq}}\right) \tag{2.27}$$

The VHDL entity file and corresponding test-bench is available from appendix D.2. In subsection 3.4.1, the design is tested and the result from the testbenches is evaluated.

2.10.3 PI Controller

In order to control $U_{2,DC}$ the voltage has to be measured and fed into the controller. The VHDL-code for this controller has been given in appendix D.3. Workgroup "Measurement and Wireless communication" provides the controller with a measurement of the voltage. Since the bit size of the incoming (digital) is still unknown, the width is specified as a generic. The bit size of the output of the controller - the duty-cycle - has a size described by subsection 2.10.1.

The PI-Controller has been programmed as a Moore machine. The FSM-diagram of the circuit is given in figure 2.19. Below the FSM-diagram, the functions of every state are explained.

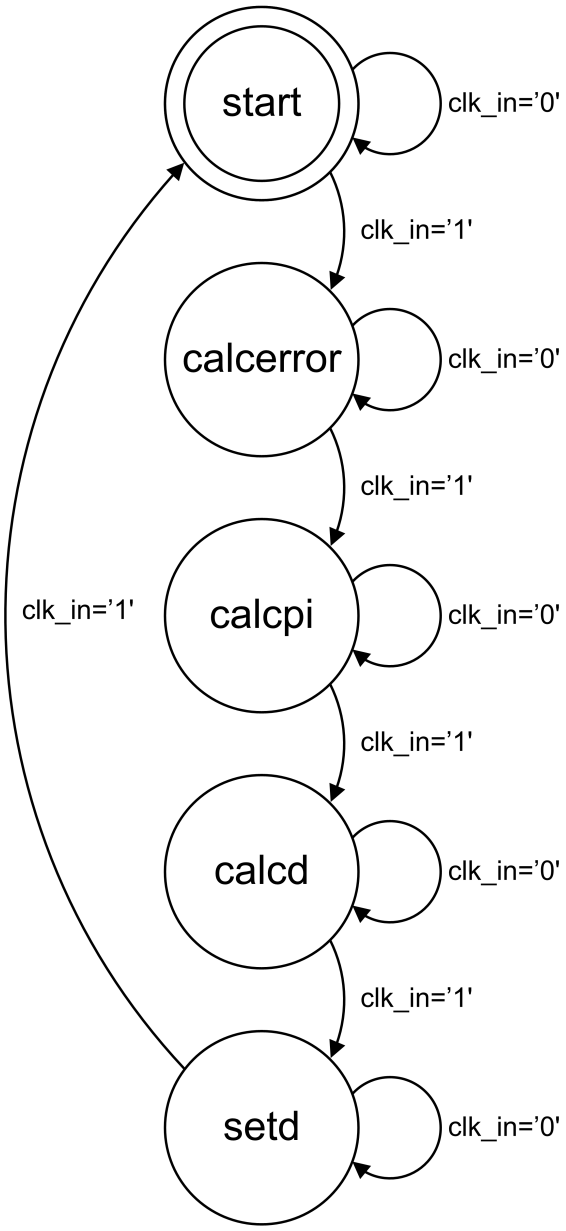


Figure 2.19: Finite state machine for the PI controller

State description for the PI-controller

- **start**

```
1          adcin <= to_integer(unsigned(data_in));
2          sp<=to_integer(unsigned(setpoint));
3          Error_Old := Error;
4          next_state<= calcerror;
```

This state reads the incoming sensor data and reads the setpoint. It also saves the previous error in `Error_Old`.

- **calcerror**

```
1          next_state<=calcpi;
2          Error <= to_integer(to_unsigned((sp-adcin)));
```

This states calculates the difference between the incoming signal - the sensor signal - and the set-point. Furthermore, it saves this integer in the variable `Error`.

- **calcpi**

```
1          next_state<=calcd;
2          p<=Kp*Error;
3          i<=Ki*(Error+Error_old);
```

`calcpi` calculates the signals in the proportional and integral branches of the controller. Since `p` is just the proportional part, the error is multiplied. And because `i` is the integral part, the old error is added to the new error.

- **calcd**

```
1  Output <=(p+i+d)/scale_factor;
2
3  if Output > max_output then
4                    Output <= max_output ;
5               end if;
6               if Output < 1 then
7                    Output <= 1;
8               end if;
9
10
11         next_state<=setd;
```

This state will calculate the output of the PID controller and saturate it when it is higher than the maximum of the duty cycle.

- **setd**

```
1          data_out<=std_logic_vector(to_unsigned(
2                         Output ,length_data_out));
3          next_state <= start;
```

This final state will put the data on the `data_out`, the output port of the entity. Furthermore it will return to the `start`-state.

2.11 Use

Implementation of the system specified in the preceding sections, will allow the next generation of wireless charging systems to operate at a fixed power set-point and and maximum efficiency. These wireless charging systems can be used for a wide range of applications.

Currently, the system is designed to work for a battery with a nominal voltage of 200V charging at 8.0kW. Misalignments must be within the range of $0.08 \leq k \leq 0.20$. The minimum voltage at the primary side must be at least 460V otherwise impedance is matched poorly. However, tuning various constants in the system will allow the system to fulfill other requirements.

3 Evaluation

3.1 Overview

Due to the fact that this work group is dependent on the progress of other groups, it was not possible to test the controllers. Instead, simulation results are presented in this chapter. This chapter starts off with the simulation results of the $U_{2,dc}$ controller followed by the α controller. After that the entire system is simulated and evaluated. The implementation of the PWM generator is then presented, followed by a assessment of the system and recommendations.

3.2 Simulations

3.2.1 Simulation of $U_{2,dc}$ controller

The topology in figure 3.1 has been simulated with different controllers. The simulation file is available in appendix E. These controllers have been trained using the genetic algorithm at 35A input voltage. This is the current at which the battery charges at 8.0kW at the worst misalignment. The values of table 2.1 are used for the buck converter. The sample and hold block has also been put into the simulation to simulate noise created by the FPGA. The FPGA can only change the duty cycle once ever 2μs due to the switching frequency. Quantization noise has not been implemented since it can be diminished by having a PWM generator with a higher resolution.

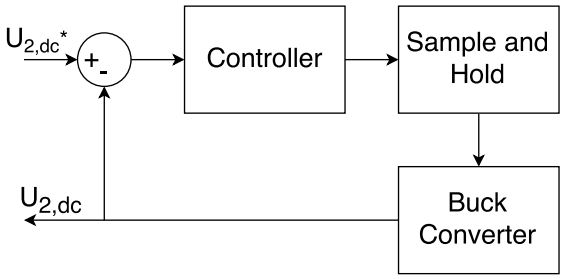


Figure 3.1: Topology of simulation.

Figure 3.2, 3.3 and 3.4 show the simulation results of the buck converter with an PI controller. The equation of the PI controller is $PI(s) = \left(-11.79 + \frac{-1.104\times10^3}{s}\right)$. The reference voltage changes from 460V (voltage at worst misalignment) to 400V. The current remains 35A, but in reality it should go down due to improvement of alignment. The initial overshoot is about 0.13% (figure 3.3) and the second overshoot is about 2.0% due to the constant current. If the error of the system (figure 3.4) is examined two things stand out. The first is that $U_{2,dc}$ is not a smooth line but a repetitive wave form during steady state. This effect is caused by the FPGA that samples and holds the duty cycle every 2μs. Similar effect is seen in [17]. The second is that the error of the system does not go to zero, but on average remains stable. This is probably due to the limitations of a linear controller on a non linear system.

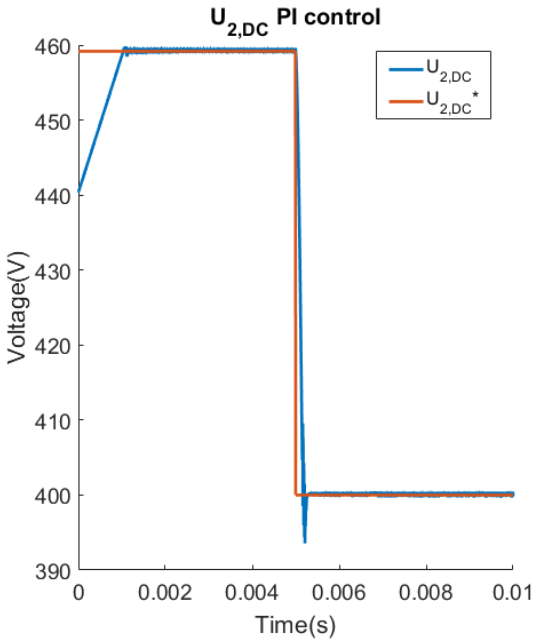


Figure 3.2: Simulations results PI controller. I_s is 35A, k = 0.2

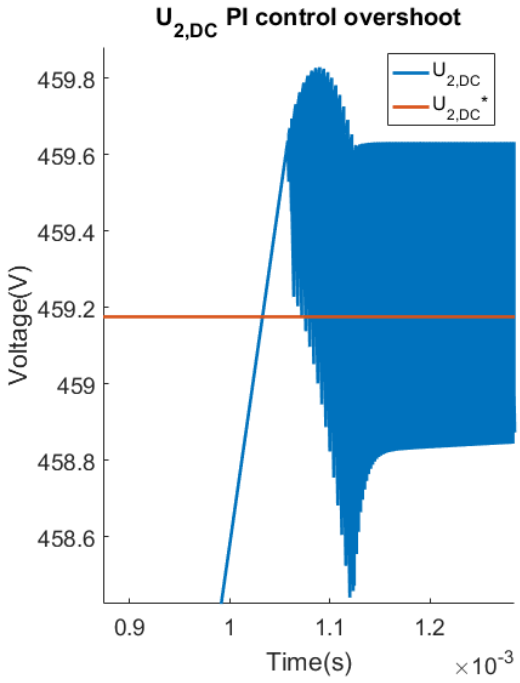


Figure 3.3: Simulations results overshoot PI controller.

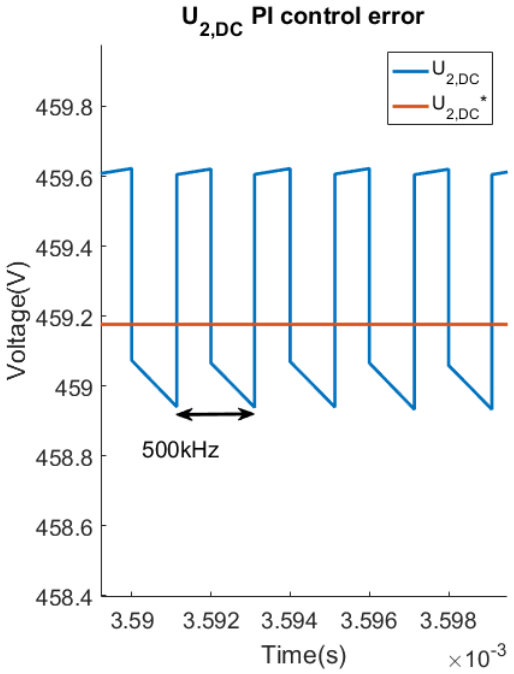


Figure 3.4: Simulations results error PI controller.

Figure 3.5 shows the the simulation results of the same PI controller, but then at the worst

alignment. The overshoot is 0.23% and the average steady state error is 0.8%.

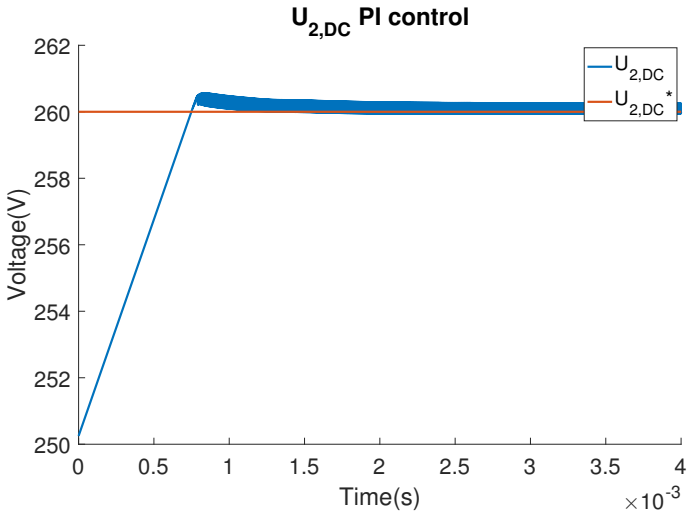


Figure 3.5: Simulations results PI controller. I_s is 35A, k = 0.08

A PID controller has also been tuned and tested. Figure 3.6 shows the simulation results. The D part also did not show any mayor improvements in the response. It could be that the constants were taken from the wrong range. The PID controller that was used is given by: $PID(s) = \left(-0.51 + \frac{-5.8\times10^2}{s} + 1.2\times10^{-3}s\right)$

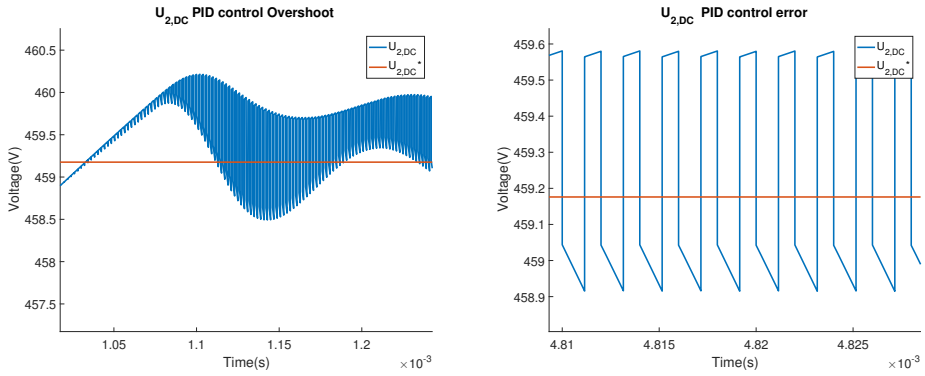


Figure 3.6: Simulations results PID controller. I_s is 35A, k = 0.2

3.2.2 Simulation of α controller

Appendix E.1 shows the topology used to simulate the controller for the inverter. Figure 3.7 shows the simulation results of the topology with a trained PI controller. The simulations shows an overshoot of 10% and an average error of 1.5% at steady state. Since there are no requirements for this controller, the results are left as they are.

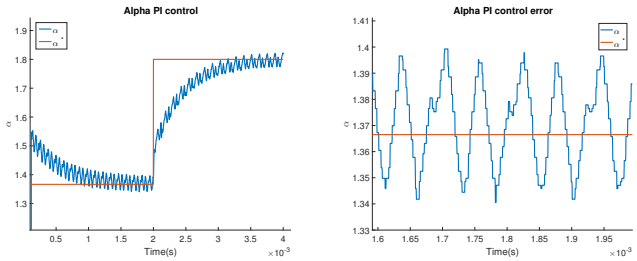


Figure 3.7: Simulations results PI controller

3.2.3 Simulation of total system

The total system has been simulated using the α and $U_{2,DC}$ controller. Both controllers were PI controllers. Figure 3.8 shows this simulation. The simulation starts off with an input current of 25A with an alignment of $k = 0.20$ (best alignment) and switches to an input current of 35A with an alignment of $k = 0.08$ (worst alignment). In both cases, the steady state power P_{bat} is approximately 8.0kW. The difference is due to losses in the system which are not compensated by $U_{1,DC}$. One thing to note is that power peaks once k is changed dramatically. This peak occurs due to the rectifying capacitor. It suddenly must lower the voltage and does this by releasing excess charge into the battery. The sudden peak in power is dangerous for the battery. Solutions to diminish this power surge, could be to lower $U_{2,DC}$ slowly, which has been done in figure 3.2.

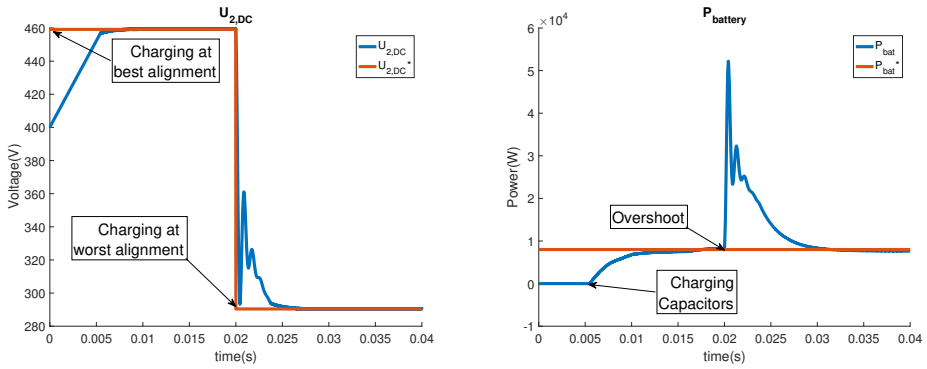


Figure 3.8: Simulations results P_{bat}

It's now possible to conclude that design requirement 1, 2,3 and 5 have been met.

3.3 Prototype

Since the buck converter has not been built there is no prototype. The controller has been implemented on an FPGA but cannot be tested since measurement data must be supplied.

The only prototype that could be tested was the PWM generator and the signal generator for the inverter.

3.4 Testing the PWM-Generator

A VHDL test-bench was written in order to test and simulate the output of the PWM-entity. The result of the test-bench were as expected. After that, the simulation was programmed into the Spartan3E and the output was measured on an analogue oscilloscope. The simulation starts with a duty cycle of 50%. Then, shortly after 10μs, a new duty cycle of 12.5% = $\frac{8}{64}$ gets written on the input ports. Shortly after, the new duty cycle is latched in. The results can be found on http://mikepieters.com/IPT.zip.

For synthesis on the Spartan3E chip, the VHDL code was modified such that the duty cycle is 50% when the device is rebooted. A duty cycle of 25% after the first latch pulse was hard-coded. This was done to simulate different duty cycles. The duty cycle has been verified using an oscilloscope. A few results of the testing are presented in figure 3.9 and 3.10.

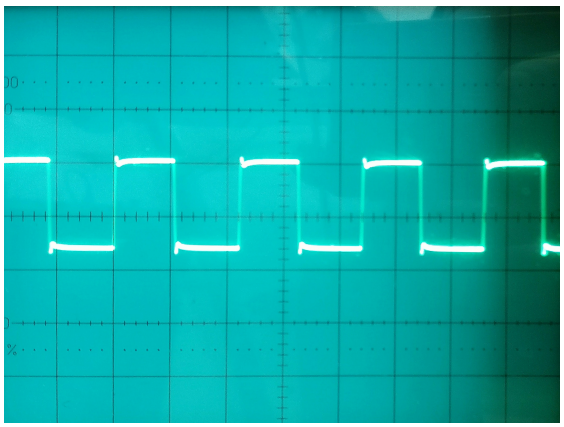


Figure 3.9: Testing PWM generator after boot with a 50% dutycycle

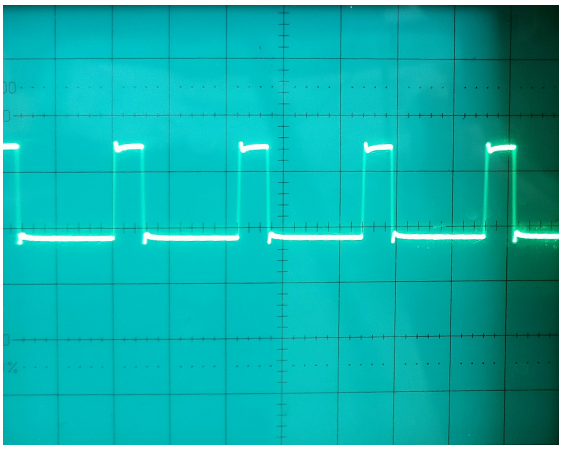


Figure 3.10: Testing PWM generator setting duty cycle of 25%

3.4.1 Testing the inverter control signal generator

The inverter signal generator was tested by using the test-bench in appendix section D.2.2. The results are difficult to show here, since the high resolution that is needed to show the detail properly, but the results can be found on http://mikepieters.com/IPT.zip.

3.5 Assessment

Concerning the buck model the expectation were higher than the final outcome. The buck model only seemed to have a representative rise time and settle time when the input current is much higher than 35A. The buck model could have saved a lot of computation time which could have improved the genetic algorithm.

The genetic algorithm worked really well. The main drawback of this method was the simulation time needed to get good results. The result of the algorithm may be a lot better when executed on a faster computer. However, the current results gave us a system that met all the requirements.

Regarding the simulation results. The verification of the maximum efficiency point yielded the expected results. The α-generator also worked really well in simulation. The controller that was chosen in the end did not bring the average error to zero due to the non linear behavior. It was, however, very close to zero and the overshoot of $U_{2,dc}$ was within the limits. The steady state error might have gone to zero when a non linear controller had been used, since literature advises the usage of it. The steady-state power output in simulation came reasonably close to the set-point. The overshoot in $U_{2,dc}$ became a lot smaller when we implemented an anti-windup method. However, when changing the coupling-coefficient k resulted in huge short peak in power output. We expect that this is due to capacitor discharging when $U_{2,dc}$ changes suddenly.

It's also a shame that the controller hasn't been physically tested. We expected a lot of quantization noise to occur, but this could not be verified. However, we managed to get the PWM-generator and the inverter signal generator working.

3.6 Next steps

The next steps in this project is the combine the work of workgroup 1,2 and 3 and connect the hardware to create this system inductive power transfer system physically. We advise to implement the both control systems, since it tackles the problem of both constant power and efficiency for wireless charging. The buck should be built and then controlled with the control system. This control system should be assessed if it is sufficient and otherwise replaced with a non linear controller. The genetic algorithm should be executed on a faster computer to produce more accurate results. The set-point for α should be transmitted using the work of the wireless-communication group.

Nomenclature

General Symbols

η	Efficiency
ω	Angular frequency of the primary source
C_p	Capacitance at primary side of the transformer
D	Duty-cycle for the PWM signal controlling the buck converter
k	Coefficient of coupling from the transformer
L_1	Inductance at primary side of the transformer
L_2	Inductance at secondary side of the transformer
M	Mutual inductance of the transformer
P_{out}	Power output of the complete IPT system
R_p	Parasitic resistance of both C_p and L_1 combined. See figure 2.2.
R_s	Parasitic resistance of both C_s and L_1 combined
R_{load}	Resistance after transformer and reactive power compensation, see figure 2.3
V_p	AC voltage at primary side of the transformer
$V_{battery}$	Nominal battery voltage
PWM	Pulse Width Modulation

Buck Symbols

C_1	Input capacitance of the buck converter
C_2	The output capacitance
D_1	Diode in the buck converter
f_{sw}	Switching frequency of the buck converter

I_s	Current at the input of the buck converter
L	The inductor in the converter
R_b	Internal resistance of the battery
R_l	Parasitic resistance of the inductor
R_{c1}	Parasitic resistance of C_1
R_{c2}	Parasitic resistance of C_2
V_b	Battery voltage

Control Algorithm Symbols

α	Phase shift between the two inverter legs
η_{rect}	Efficiency of the rectifier
L_p	Inductance at primary side of the transformer
L_s	Inductance at secondary side of the transformer
R_{ac}	Resistance in front of the rectifier, see figure 2.6
R_{buckin}	Resistance after the rectifier, in front of the buck converter
R_{load}	Resistance in front of the rectifier, see figure 2.5.
$U_{1,dc}$	Voltage in front of the inverter, see figure 2.5
$U_{2,dc}$	Voltage in front of the rectifier, see figure 2.5

Controller Symbols

e_{ss}	Steady state error
K_p	Proportional term of the PI controller
T_i	Integral term of the PI controller
D	Derivative
I	Integral
P	Proportional

FPGA Implementation Symbols

f_{clk}	Clock frequency of the Papilio One development board
f_{PWM}	Frequency of the generated PWM Signal

k Number of inputs needed to define the duty-cycle

n Number of quantization levels

A Calculation maximum efficiency

A.1 Calculation efficiency

$Z_{total} = (R_s + R_L + Z_s) \parallel Z_m + Z_p + R_p = \frac{Z_m(R_s+R_L+Z_s)}{Z_m+R_s+R_L+Z_s} + Z_P + R_P = R_P + j\omega(L_p - M) - j\omega L_p + \frac{j\omega M(R_s+R_L-j\omega M)}{R_s+R_L}$

Take $L_s = L_P$.

$= R_p + \frac{\omega^2 m^2}{R_s+R_L}$

$I_p = \frac{V_p}{Z_{total}} = \frac{V_p}{R_p + \frac{\omega^2 M^2}{R_s+RL}} = \frac{V_P(R_s+R_L)}{(R_s+R_L)R_P+\omega^2 M^2}$

$I_L = \frac{I_P Z_M}{Z_M+Z_s+R_s+R_L} = \frac{V_P(R_s+R_L)}{(R_s+R_L)R_P+\omega^2 M^2} \frac{j\omega M}{j\omega M+Z_s+R_s+R_L} = \frac{V_P(R_s+R_L)}{(R_s+R_L)R_P+\omega^2 M^2} \frac{j\omega M}{R_s+R_L} = \frac{jV_P\omega M}{(R_s+R_L)R_p+\omega^2 M^2}$

$P_{out} = \frac{V_p^2 \omega^2 M^2 R_L}{[(R_s+R_L)R_p+\omega^2 M^2]^2}$

$P_{in} = \frac{V_p^2}{R_p + \frac{\omega^2 M^2}{R_s+R_L}} = \frac{(R_s+R_L)V_p^2}{(R_s+R_L)R_p+\omega^2 M^2}$

$\eta = \frac{P_{out}}{P_{in}} = \frac{V_p^2 \omega^2 M^2 R_L}{[(R_s+R_L)R_P+\omega^2 M^2]} \frac{(R_s+R_L)R_p+\omega^2 M^2}{(R_s+R_L)V_p^2} = \frac{\omega^2 M^2 R_L}{(R_s+R_L)^2 R_p+\omega^2 M^2(R_s+R_L)} = \frac{\omega^2 M^2 R_L}{R_p(R_s^2+R_L^2+2R_sR_L)+\omega^2 M^2 R_s+\omega^2 M^2 R_L}$

$\frac{\mathrm{d}\eta}{\mathrm{d}R_L} = \frac{[(R_s+R_L)^2 R_p+\omega^2 M^2 R_s+\omega^2 M^2 R_L]\omega^2 M^2 - \omega^2 M^2 R_L(2R_sR_p+2R_LR_p+\omega^2 M^2)}{[(R_s+R_L)^2 R_p+\omega^2 M^2(R_s+R_L)]^2} = 0$

Matlab yields:

$R_L = \frac{1}{R_p}\sqrt{RpR_s(M^2\omega^2 + R_pR_s)} = \frac{1}{R_p}\sqrt{RpR_sM^2\omega^2 + R_p^2R_s^2} \approx \sqrt{M^2\omega^2\frac{R_s}{R_p}} = \omega M\sqrt{\frac{R_s}{R_p}}$

A.2 Calculation state space system

Switch open:

$\mathbf{V_{C1}}: \frac{dV_{c1}}{dt} = \frac{1}{C_1} I_s$

Appendix A. Calculation maximum efficiency

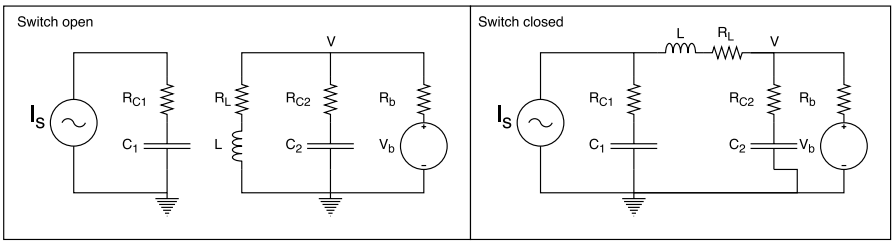


Figure A.1: Buck converter when switch is open or closed.

$\mathbf{V}: I_L = I_b + I_{C2} = \frac{V-V_{C2}}{R_{C2}} + \frac{V-V_b}{R_b} = V(\frac{1}{R_{C2}} + \frac{1}{R_b}) - \frac{V_b}{R_b} - \frac{V_{C2}}{R_{C2}}$

$I_L + \frac{V_b}{R_b} + \frac{V_{C2}}{R_{C2}} = \frac{V}{R_{C2} \parallel R_b} \rightarrow V = R_{C2} \parallel R_b(I_L + \frac{V_b}{R_b} + \frac{V_{C2}}{R_{C2}})$

$\mathbf{I_L}: \frac{dI_L}{dt} = \frac{1}{L}(-R_L I_L - V) = \frac{1}{L}(-R_L I_L - R_{C2} \parallel R_b(I_L + \frac{V_b}{R_b} + \frac{V_{C2}}{R_{C2}}))$

$\mathbf{V_{C2}}: \frac{dV_{C2}}{dt} = \frac{1}{C_2}(I_L - I_b) = \frac{1}{C_2}(I_L - \frac{V-V_b}{R_b}) = \frac{1}{C_2}(I_L - \frac{R_{C2} \parallel R_b}{R_b}(I_L + \frac{V_b}{R_b} + \frac{V_{C2}}{R_{C2}}) + \frac{V_b}{R_b})$

$$\mathbf{X}' = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \frac{-1}{L}(R_L + \frac{R_{C2}R_b}{R_{C2}+R_b}) & \frac{-R_b}{L(R_{C2}+R_b)} \\ 0 & \frac{R_b}{C_2(R_{C2}+R_b)} & \frac{-1}{C_2(R_{C2}+R_b)} \end{bmatrix} \mathbf{X} + \begin{bmatrix} \frac{I_s}{C_1} \\ \frac{-R_{C2}V_b}{L(R_{C2}+R_b)} \\ \frac{V_b}{C_2(R_{C2}+R_b)} \end{bmatrix}$$

$\mathbf{Y}: Y = R_{C1}(I_s - I_L) + V_{C1}$

Switch closed:

$\mathbf{V_{C1}}: \frac{dV_{c1}}{dt} = \frac{1}{C_1}(I_s - I_L)$

$\mathbf{I_L}: \frac{dI_L}{dt} = \frac{1}{L}(V_{C1} + R_{C1}(I_s - I_L) - I_L R_L - V) = \frac{1}{L}(V_{C1} - (R_{C1} + R_{C2} \parallel R_b + R_L))I_L - \frac{R_{C2} \parallel R_b}{R_{C2}} V_{C2} + R_{C1} I_s - \frac{R_{C2} \parallel R_b}{R_b} V_b$

$\mathbf{V_{C2}}: \frac{dV_{C2}}{dt} = \frac{1}{C_2}(I_L - I_b) = \frac{1}{C_2}(I_L - \frac{V-V_b}{R_b}) = \frac{1}{C_2}(I_L - \frac{R_{C2} \parallel R_b}{R_b}(I_L + \frac{V_b}{R_b} + \frac{V_{C2}}{R_{C2}}) + \frac{V_b}{R_b})$

$$\mathbf{X}' = \begin{bmatrix} 0 & \frac{-1}{C_1} & 0 \\ \frac{1}{L} & \frac{-1}{L}(R_{C1} + R_L + R_{C2} \parallel R_b) & \frac{-R_b}{L(R_{C2}+R_b)} \\ 0 & \frac{R_b}{C_2(R_{C2}+R_b)} & \frac{-1}{C_2(R_{C2}+R_b)} \end{bmatrix} \mathbf{X} + \begin{bmatrix} \frac{I_s}{C_1} \\ \frac{1}{L}(I_s R_{C1} - \frac{-R_{C2}V_b}{R_{C2}+R_b}) \\ \frac{V_b}{C_2(R_{C2}+R_b)} \end{bmatrix}$$

$\mathbf{Y}: Y = R_{C1} I_s + V_{C1}$

Average state space

$$\mathbf{X_{avg}}' = \begin{bmatrix} 0 & \frac{-d}{C_1} & 0 \\ \frac{d}{L} & \frac{-1}{L}(R_L + R_{C1}d + R_{C2} \parallel R_b) & -\frac{R_b}{L(R_{C2}+R_b)} \\ 0 & \frac{R_b}{C_2(R_{C2}+R_b)} & \frac{-1}{C_2(R_{C2}+R_b)} \end{bmatrix} \mathbf{X_{avg}} + \begin{bmatrix} \frac{I_s}{C_1} \\ \frac{1}{L}(I_s R_{C1} d - \frac{-R_{C2}V_b}{R_{C2}+R_b}) \\ \frac{V_b}{C_2(R_{C2}+R_b)} \end{bmatrix}$$

$$\mathbf{Y}: Y = \begin{bmatrix} 1 & R_{C1}d & 0 \end{bmatrix} + I_s R_{C1}$$

B Maximum Efficiency Point Simulation

The following simulation has been executed to verify whether R_{load} is indeed equal to $\omega M = \omega k L_{transformer}$,

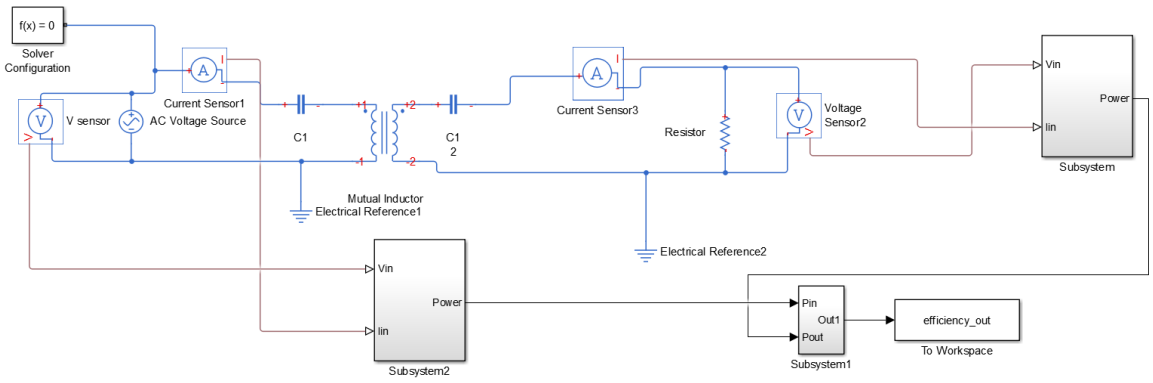


Figure B.1: The simulation done to verify the maximum efficiency point.

```
clear all;

%Set some constants
f=85000;
w=f*2*pi;
Ltransformer=200e-6;
Ctransformer=1/(w^2*Ltransformer);

%Create the sweeping variables
resistance=[8:0.5:25];
k_values=[0.11:0.02:0.19]
results=zeros(length(resistance),length(k_values))
```

```
h = waitbar(0/length(resistance),'Simulating results')

%Sweep over numerous resistances for different values of k
 for j=1:length(k_values)
     k=k_values(j);
 for n=1:length(resistance)
    waitbar(n / length(resistance),h,strcat('Number of k:', '[',num2str(j),
    '/',num2str(length(k_values)),']'));
    R= resistance(n);
    disp(R);
    sim('effienciency');
    results(n,j)=efficiency_out.Data(2);

 end

 end

 %Get for every k the maximum efficiency point
 a=1:length(k_values);
 b=1:length(k_values);

 for m=1:length(k_values)
    [a(m),b(m)]=max(results(:,m));
 end


%Plot the results
for n=1:length(k_values)
plot(resistance,results(:,n),'DisplayName',strcat(
                        'k=',num2str(k_values(n))));
hold on;
end
hold off
title('Resistance vs Efficiency')
xlabel('Resistance')
ylabel('Efficiency')
legend('show')
```

C Matlab code genetic algorithm

C.1 Main script

```
clear all
close

%Range
Kp_min = 0; Kp_max = 1e1;
Ti_min = 0; Ti_max = 1e3;
Kd_min = 0; Kd_max = 1e-2;
sigKp_max = 1e-1;
sigTi_max = 1e-2;
sigKd_max = 1e-3;
popsize = 1000; iterations = 60;

%Evoluntionary Strategy
%Start population
[sp_kp, sp_ti, sp_kd, sp_skp, sp_sti, sp_skd] = StartPopulation(popsize
, Kp_min, Kp_max, Ti_min, Ti_max, Kd_min, Kd_max, sigKp_max, sigTi_max
, sigKd_max);

%Fitness of start population
disp('Assessing Fitness');
fit = FitnessFunction(sp_kp, sp_ti, sp_kd);
disp('Done');
startpop = [sp_kp, sp_ti, sp_kd, sp_skp, sp_sti, sp_skd, fit];
clear fit sp_kp sp_ti sp_skp sp_sti;  %Remove variables

currentpop = startpop;

```

```
28
29 disp('Starting');
30 %Iteration
31 bestsolution = zeros(1,4);
32 for i = 1:iterations
33     %Generate offspring
34     [offKp, offTi, offKd, offsKp, offSTi, offsKd] = NewSolution(
35     Kp_min, Kp_max, Ti_min, Ti_max, Kd_min, Kd_max, sigKp_max,
36     sigTi_max, sigKd_max, currentpop);
37     %Rate fitness
38     offFit = FitnessFunction(offKp, offTi, offKd);
39
40     %Reorder offspring
41     offspring = [offKp, offTi, offKd, offsKp, offSTi, offsKd, offFit];
42     %Recreate population
43
44    %Statistical purposes
45     nextpop = [currentpop; offspring];  %Append offspring to current population
46     nextpop = sortrows(nextpop, 7);     %Sort t.b. examined population by fitness
47
48
49     %Keep track of best solution
50     if(nextpop(end, 7) > bestsolution(1,4))
51         bestsolution = [nextpop(end,1), nextpop(end,2), nextpop(end,3), nextpop(end,7)];
52     end
53
54     currentpop = nextpop(popsize+1:end,:);
55     disp(i);
56 end
57
58
59 K = bestsolution(1); T = bestsolution(2);
```

C.2 Start population

```
1 function [ Kp, Ti, sigKp, sigTi ] =
2 StartPopulation( popsize, Kp_min, Kp_max, Ti_min, Ti_max, sigKp_max, sigTi_max)
3 %This function generates a start population
4
5 %Constants
6 Kp = transpose(linspace(Kp_min, Kp_max, popsize^0.5)); %Constant Kp
7 Kp = kron(Kp, ones(popsize^0.5,1));
```

```
Ti = transpose(linspace(Ti_min, Ti_max, popsize^0.5)); %Constant Ti
Ti = repmat(Ti,popsize^0.5,1);
sigKp = sigKp_max*rand(popsize,1);      %Sigma Kp
sigTi = sigTi_max*rand(popsize,1);      %Sigma Ti

end
```

C.3 Fitness function

```
function [ fitness ] = FitnessFunction(Kp, Ti, Kd )
%Fitness : vector of fitness function
%Kp : Vector of Kp constants
%Ti : Vector of Ti constants
%Kp must be of same length as Ti

close_system('System', 0);
run('Buck_design_constants');

Is = 35;
V_ref = ((pi^2 / 8)*2*pi*85000 * 200e-6*0.2 * 8000)^0.5;

spmd
    load_system('System');
    evalin('base','Buck_design_constants');
    assignin('base','k_p',Kp);
    assignin('base','t_i',Ti);
    assignin('base','k_d',Kd);
    set_param('System','MaskedZcDiagnostic','none');
    set_param('System','IgnoredZcDiagnostic','none');
end
simout(1:length(Kp)) = Simulink.SimulationOutput;

%Fitness function
disp('Starting fitness');
parfor i = 1:length(Kp)
    %Set parameters
    assignin('base','i',i);
    evalin('base','K = k_p(i);');
    evalin('base','T = t_i(i);');
    evalin('base','D = k_d(i);')


```

```
    try
        simout(i) = sim('System', 'SaveState','on','StateSaveName','xout',...
            'SaveOutput','on','OutputSaveName','yout','SaveFormat','Dataset');
    catch
    end
end

fitness = zeros(length(Kp),1);

for i = 1:length(Kp)
    try
        VC1_out = simout(i).get('yout').get('VC1').Values.Data;
        overshoot = (max(VC1_out(0:round(end/4))) - V_ref)/V_ref;
        err = (mean(VC1_out(2*end/5 : end)) - V_ref)/V_ref;

        if(abs(VC1_out(end) - V_ref)/V_ref < 0.05)
            fit_err = min([0.5 1/(200*err)]);
            fit_overshoot = min([0.5 1/(200*overshoot)]);
            fitness(i) = fit_err + fit_overshoot;
        else
            fitness(i) = 0;
        end
    catch
            fitness(i) = 0;
    end
end

close_system('System', 0);
return;

end
```

C.4 Offspring

```
function [newKp, newTi, newKd, newsKp, newSTi, newsKd] =
NewSolution(Kp_min, Kp_max, Ti_min, Ti_max, Kd_min, ...
Kd_max, sigKp_max, sigTi_max, sigKd_max, currentpop)

%Splitting for parallelization
oldKp = currentpop(:,1);
oldTi = currentpop(:,2);
oldKd = currentpop(:,3);
```

```
9  oldsKp = currentpop(:,4);
10 oldsTi = currentpop(:,5);
11 oldsKd = currentpop(:,6);
12
13 %Initialize variables
14 popsize = size(currentpop, 1);
15 newKp = zeros(popsize, 1);
16 newTi = zeros(popsize, 1);
17 newKd = zeros(popsize, 1);
18 newsKp = zeros(popsize, 1);
19 newSTi = zeros(popsize, 1);
20 newsKd = zeros(popsize, 1);
21
22 parfor i = 1:length(currentpop)
23     %Determine variables
24     Kp = oldKp(i) + normrnd(0, oldsKp(i));
25     Ti = oldTi(i) + normrnd(0, oldsTi(i));
26     Kd = oldKd(i) + normrnd(0, oldsKd(i));
27
28     newsKp(i) = abs(oldsKp(i) + normrnd(0, sigKp_max));
29     newSTi(i) = abs(oldsTi(i) + normrnd(0, sigTi_max));
30     newsKd(i) = abs(oldsKd(i) + normrnd(0, sigKd_max));
31
32     %Setting Limit Kp
33     if(Kp > Kp_max)
34         newKp(i) = Kp_max;
35     elseif(Kp < Kp_min)
36         newKp(i) = Kp_min;
37     else
38         newKp(i) = Kp;
39     end
40
41     %Setting limit Ti
42     if(Ti > Ti_max)
43         newTi(i) = Ti_max;
44     elseif(Ti < Ti_min)
45         newTi(i) = Ti_min;
46     else
47         newTi(i) = Ti;
48     end
49
50     %Setting limit Kd
```

```
51      if(Kd > Kd_max)
52          newKd(i) = Kd_max;
53      elseif(Kp < Kp_min)
54          newKd(i) = Kd_min;
55      else
56          newKd(i) = Kd;
57      end
58
59  end
```

D VHDL Codes

D.1 VHDL code for the PWM Generator

D.1.1 VHDL Entity

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6
7  entity PWM is
8
9  generic(
10     clk_freq    : integer :=32_000_000;
11     pwm_freq : integer :=500_000;
12     bits_res : integer :=6 -- 2^6=64
13 );
14
15 port(
16 CLK : in STD_LOGIC;
17 RESET : in STD_LOGIC;
18 LATCH : in STD_LOGIC;
19 dutycycle_IN : in STD_LOGIC_VECTOR(bits_res - 1 downto 0);
20 PWM_OUT : out STD_LOGIC
21 );
22
23 end PWM;
24
25 architecture Behavioral of PWM is
```

```
26
27 CONSTANT clocks_per_PWM : integer:= clk_freq/pwm_freq;
28 SIGNAL count : integer range 0 to clocks_per_PWM+1;
29 SIGNAL time_high : integer range 0 to clocks_per_PWM:=32;
30
31 begin
32
33 process(CLK,RESET)
34 begin
35 if(RESET='1') then
36     count<=0;
37     pwm_out<='0';
38 elsif(rising_edge(CLK)) THEN
39         if(LATCH='1') then
40             time_high<=conv_integer(dutycycle_IN);
41             count<=0;
42             PWM_OUT<='0';
43         end if;
44
45         if(count=clocks_per_PWM) then
46             count<=0;
47         else
48             count <= count + 1;
49         end if;
50
51         if(count=1) then
52             PWM_OUT<='1';
53         elsif(count=time_high) then
54             PWM_OUT<='0';
55         end if;
56 end if;
57
58 end process;
59
60 end Behavioral;
```

D.1.2 VHDL Testbench

```
1
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.ALL;
4
5 -- Uncomment the following library declaration if using
```

```
6  -- arithmetic functions with Signed or Unsigned values
7  --USE ieee.numeric_std.ALL;
8
9  ENTITY tb_pwm IS
10 END tb_pwm;
11
12 ARCHITECTURE behavior OF tb_pwm IS
13
14     -- Component Declaration for the Unit Under Test (UUT)
15
16     COMPONENT PWM
17     PORT(
18          CLK : IN  std_logic;
19          RESET : IN  std_logic;
20          LATCH : IN  std_logic;
21          dutycycle_IN : IN  std_logic_vector(5 downto 0);
22          PWM_OUT : OUT  std_logic
23         );
24     END COMPONENT;
25
26
27    --Inputs
28    signal CLK : std_logic := '0';
29    signal RESET : std_logic := '0';
30    signal LATCH : std_logic := '0';
31    signal dutycycle_IN : std_logic_vector(5 downto 0)
32     := (others => '0');
33
34     --Outputs
35    signal PWM_OUT : std_logic;
36
37    -- Clock period definitions
38    constant CLK_period : time := 31.25 ns;
39
40 BEGIN
41
42     -- Instantiate the Unit Under Test (UUT)
43    uut: PWM PORT MAP (
44           CLK => CLK,
45           RESET => RESET,
46           LATCH => LATCH,
47           dutycycle_IN => dutycycle_IN,
```

```
48            PWM_OUT => PWM_OUT
49          );
50
51     -- Clock process definitions
52     CLK_process :process
53     begin
54          CLK <= '0';
55          wait for CLK_period/2;
56          CLK <= '1';
57          wait for CLK_period/2;
58     end process;
59
60
61     -- Stimulus process
62     stim_proc: process
63     begin
64        -- hold reset state for 100 ns.
65        wait for 100 ns;
66
67        wait for CLK_period*10;
68
69          wait for 10000 ns;
70           dutycycle_IN<="001000";
71          wait for 100 ns;
72          LATCH<='1';
73          wait for 100 ns;
74          LATCH<='0';
75
76
77        wait;
78     end process;
79
80  END;
```

D.2 VHDL code for the inverter control signals

D.2.1 VHDL Entity

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_UNSIGNED.ALL;
4
5
```

```
6  entity INVCONTROL is
7
8  generic(
9      clk_freq    : integer :=32_000_000;
10     control_freq : integer :=85_000; -- Frequency of the full cycle
11     bits_res : integer := 8 -- 2^8=512;
12 );
13
14 port(
15 CLK : in STD_LOGIC;
16 RESET : in STD_LOGIC;
17 LATCH : in STD_LOGIC;
18 alpha_IN : in STD_LOGIC_VECTOR(bits_res - 1 downto 0);
19 CONT1_OUT : out STD_LOGIC;
20 CONT2_OUT : out STD_LOGIC
21 );
22
23 end INVCONTROL;
24
25 architecture Behavioral of INVCONTROL is
26
27 CONSTANT clocks_per_cycle : integer:= 376;
28             --clk_freq/pwm_freq;
29 CONSTANT clocks_per_pulse : integer:= 188; -- maximum number of
30 clockcycles per inverter leg (alpha maximum)
31 SIGNAL count : integer range 0 to clocks_per_cycle; -- init counter
32 SIGNAL time_high : integer range 0 to clocks_per_pulse:=188;
33
34
35 begin
36
37 process(CLK,RESET)
38 begin
39 if(RESET='1') then
40     count<=0;
41     CONT1_OUT<='0';
42     CONT2_OUT<='0';
43 elsif(rising_edge(CLK)) THEN
44         if(LATCH='1') then -- Latches in new alpha
45             time_high<=conv_integer(alpha_IN);
46         end if;
47
```

```
48          if(count=clocks_per_cycle) then
49              count<=0;
50          else
51              count <= count + 1;
52          end if;
53
54          if(count=0) then
55              CONT1_OUT<='1';
56              CONT2_OUT<='0';
57          end if;
58
59          if(count=time_high) then
60              CONT1_OUT<='0';
61              CONT2_OUT<='0';
62          end if;
63
64          if(count=clocks_per_pulse ) then
65              CONT1_OUT<='0';
66              CONT2_OUT<='1';
67          end if;
68           if(count=time_high + clocks_per_pulse ) then
69              CONT1_OUT<='0';
70              CONT2_OUT<='0';
71          end if;
72  end if;
73
74  end process;
75
76  end Behavioral;
```

D.2.2 VHDL Testbench

```
1
2   LIBRARY ieee;
3   USE ieee.std_logic_1164.ALL;
4
5   ENTITY tb IS
6   END tb;
7
8   ARCHITECTURE behavior OF tb IS
9
10      -- Component Declaration for the Unit Under Test (UUT)
11
```

```
12      COMPONENT INVCONTROL
13      PORT(
14           CLK : IN  std_logic;
15           RESET : IN  std_logic;
16           LATCH : IN  std_logic;
17           alpha_IN : IN  std_logic_vector(7 downto 0);
18           CONT1_OUT : OUT  std_logic;
19           CONT2_OUT : OUT  std_logic
20          );
21      END COMPONENT;
22
23
24     --Inputs
25     signal CLK : std_logic := '0';
26     signal RESET : std_logic := '0';
27     signal LATCH : std_logic := '0';
28     signal alpha_IN : std_logic_vector(7 downto 0) := (others => '0');
29
30      --Outputs
31     signal CONT1_OUT : std_logic;
32     signal CONT2_OUT : std_logic;
33
34     -- Clock period definitions
35     constant CLK_period : time := 31.25 ns;
36
37 BEGIN
38
39      -- Instantiate the Unit Under Test (UUT)
40     uut: INVCONTROL PORT MAP (
41            CLK => CLK,
42            RESET => RESET,
43            LATCH => LATCH,
44            alpha_IN => alpha_IN,
45            CONT1_OUT => CONT1_OUT,
46            CONT2_OUT => CONT2_OUT
47          );
48
49     -- Clock process definitions
50     CLK_process :process
51     begin
52          CLK <= '0';
53          wait for CLK_period/2;
```

```
54          CLK <= '1';
55          wait for CLK_period/2;
56      end process;
57
58
59      -- Stimulus process
60      stim_proc: process
61      begin
62         -- hold reset state for 100 ns.
63         wait for 100 ns;
64
65          wait for CLK_period*10;
66
67           --Testing various Alpha's and latching them in
68          wait for 10000 ns;
69            alpha_IN <="00100000";
70            wait for 10000 ns;
71            LATCH <= '1';
72            wait for 100 ns;
73            LATCH <= '0';
74
75            wait for 10000 ns;
76            alpha_IN <="00010000";
77            wait for 10000 ns;
78            LATCH <= '1';
79            wait for 100 ns;
80            LATCH <= '0';
81
82             wait for 10000 ns;
83            alpha_IN <="00001000";
84            wait for 10000 ns;
85            LATCH <= '1';
86            wait for 100 ns;
87            LATCH <= '0';
88
89            wait for 10000 ns;
90            alpha_IN <="00000010";
91            wait for 10000 ns;
92            LATCH <= '1';
93            wait for 100 ns;
94            LATCH <= '0';
95
```

```
96            wait for 10000 ns;
97            alpha_IN <="00100000";
98            wait for 10000 ns;
99            LATCH <= '1';
100           wait for 100 ns;
101           LATCH <= '0';
102
103       wait;
104   end process;
105
106 END;
```

D.3 VHDL code for the PI Controller

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.NUMERIC_STD.ALL;
5
6
7  entity PIController is
8      generic(
9          length_data_in : integer := 16;
10         length_data_out : integer := 6;
11         length_setpoint_in : integer := 16;
12         max_output : integer :=64; --2^6
13         scale_factor: integer :=1024
14         );
15
16     port(
17     data_in : in STD_LOGIC_VECTOR(length_data_in - 1 to 0);
18     setpoint : in STD_LOGIC_VECTOR(length_setpoint_in - 1 to 0);
19     data_out : out STD_LOGIC_VECTOR(length_data_out - 1 to 0);
20     clk_in : in STD_LOGIC
21     );
22
23 end PIController;
24
25
26
27 architecture Behavioral of PIController is
28     type states is (start,calcerror,calcpi,calcd,setd);
```

```
29
30      signal state, nextstate : statetypes := reset;
31      signal Kp : integer:=10;
32      signal Ki : integer:=10;
33      signal Output, Error : integer;
34      signal sp : integer:=10;
35      signal p,i,d : integer;
36      signal adcin : integer;
37
38  begin
39
40
41  process(clk_in,state)
42
43  variable Error_Old : integer := 0;
44
45  begin
46
47  IF clk_in'EVENT AND clk_in='1' THEN
48                  state <= next_state;
49  END IF;
50
51  case state is
52
53      -- Get data, get SP, save old error
54      when start =>
55          adcin <= to_integer(unsigned(data_in));
56          sp<=to_integer(unsigned(setpoint));
57          Error_Old := Error;
58          next_state<= calcerror;
59
60      -- Calcualte new error
61      when calcerror =>
62          next_state<=calcpi;
63          Error <= to_integer(to_unsigned((sp-adcin)));
64
65      -- Calculate p and i output values
66      when calcpi =>
67          next_state<=calcd;
68          p<=Kp*Error;
69          i<=Ki*(Error+Error_old);
70
```

```
71      -- calculate controller output, and saturate it
72      when calcd =>
73          Output <=(p+i+d)/scale_factor;
74
75          if Output > max_output then
76                    Output <= max_output ;
77              end if;
78              if Output < 1 then
79                    Output <= 1;
80              end if;
81
82
83          next_state<=setd;
84
85      -- put the output of the controller on the output of the entity
86      when setd=>
87
88          data_out<=std_logic_vector(to_unsigned(
89          Output ,length_data_out));
90          next_state <= start;
91
92      end case;
93
94  end process;
95
96  end Behavioral;
```

E Simulink Simulations

E.1 Control topology at primary side

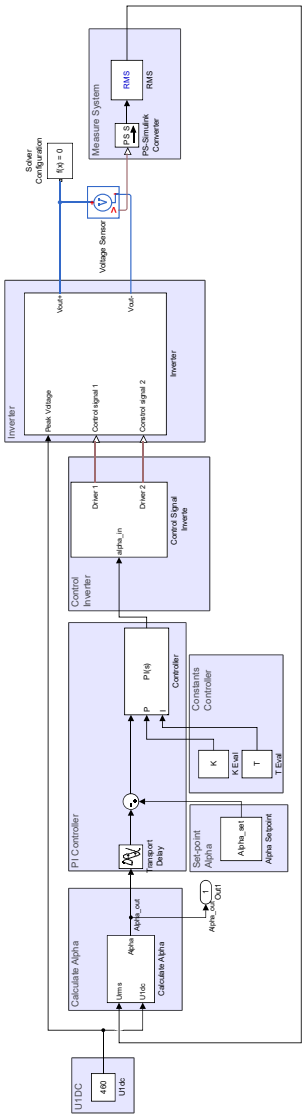


Figure E.1: Control topology at primary side

E.2 Control topology at secondary side

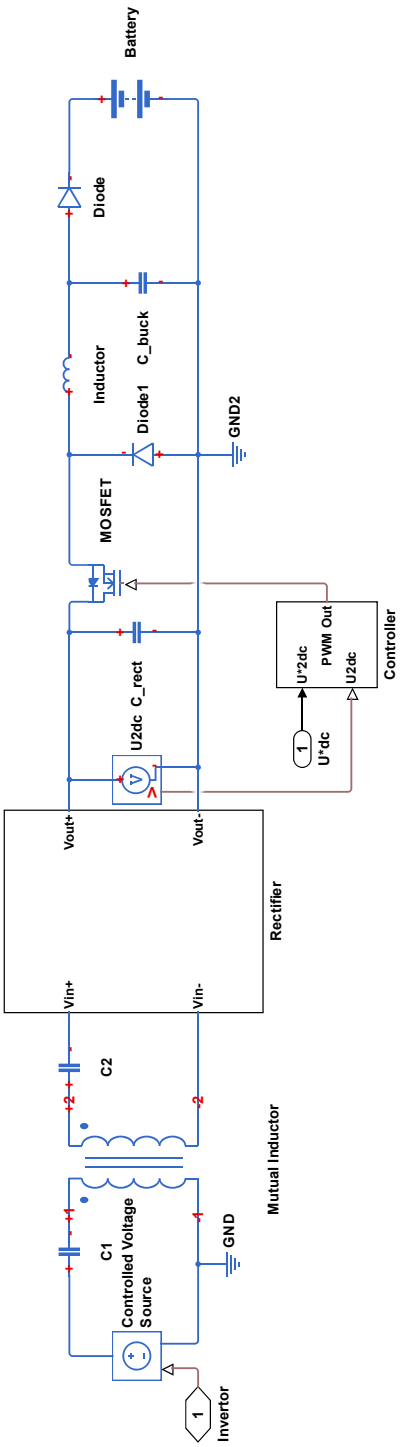


Figure E.2: Control topology at secondary side

Bibliography

[1] T. Ogretmen and N. Aboulhorma, "Misalignment Tolerant IPT Systems, Buck Design," tech. rep., Delft University of Technology, 01 2017.

[2] "Tesla home charging installation." https://www.tesla.com/en_GB/support/home-charging-installation. (Accessed on 06/12/2017).

[3] C. Simpson, *LM2576,LM3420,LP2951,LP2952 Battery Charging*. Texas Instruments, 2011.

[4] B. Ferreira, *The principles of electronic and electromechanic power conversion : a systems approach*. Hoboken, New Jersey: Wiley/IEEE, 2014.

[5] R. L. Steigerwald, "A comparison of half-bridge resonant converter topologies," *IEEE Transactions on Power Electronics*, vol. 3, pp. 174–182, Apr 1988.

[6] R. Bosshard, J. W. Kolar, J. Mühlethaler, I. Stevanović, B. Wunsch, and F. Canales, "Modeling and eta - $alpha$ -pareto optimization of inductive power transfer coils for electric vehicles," *IEEE Journal of Emerging and Selected Topics in Power Electronics*, vol. 3, pp. 50–64, March 2015.

[7] R. Mammano, "Switching Power Supply Topology Voltage Mode vs. Current Mode ," tech. rep., Unitrode IC Corporation, 1999.

[8] T. Diekhans and R. W. D. Doncker, "A dual-side controlled inductive power transfer system optimized for large coupling factor variations and partial load," *IEEE Transactions on Power Electronics*, vol. 30, pp. 6320–6328, Nov 2015.

[9] R. Bosshard, J. W. Kolar, and B. Wunsch, "Control method for inductive power transfer with high partial-load efficiency and resonance tracking," in *2014 International Power Electronics Conference (IPEC-Hiroshima 2014 - ECCE ASIA)*, pp. 2167–2174, May 2014.

[10] F. González-Longatt, "Circuit based battery models: A review." http://cs.marlboro.edu/courses/spring2009/tutorials/alec/April_15.attachments/A2006-14-1.pdf, 2006. (Accessed on 06/08/2017).

[11] S. Seshagiri, E. Block, I. Larrea, and L. Soares, "Optimal pid design for voltage mode control of dc-dc buck converters," in *2016 Indian Control Conference (ICC)*, pp. 99–104, Jan 2016.

[12] E. Block, "Linear voltage and current mode control for the dc-dc buck converter," 2015.

[13] C. Alexander, *Fundamentals of electric circuits*. New York, NY: McGraw-Hill, 2013.

[14] K. P. . R. H. A. Packard, "Me 132, dynamic systems and feedback class notes." https://jagger.berkeley.edu/~pack/me132/AllNotes.pdf. (Accessed on 06/13/2017).

[15] G. Franklin, *Feedback control of dynamic systems*. Harlow: Pearson Education Limited, 2014.

[16] M. Negnevitsky, *Artificial intelligence : a guide to intelligent systems*. Harlow, England New York: Addison Wesley/Pearson, 2011.

[17] M. Truntič and M. Milanovič, "Voltage and current-mode control for a buck-converter based on measured integral values of voltage and current implemented in fpga," *IEEE Transactions on Power Electronics*, vol. 29, pp. 6686–6699, Dec 2014.

[18] P. Omer, J. Kumar, and B. S. Surjan, "Design of robust pid controller for buck converter using bat algorithm," in *2016 IEEE 1st International Conference on Power Electronics, Intelligent Control and Energy Systems (ICPEICES)*, pp. 1–5, July 2016.

[19] "Papilio one hardware." http://papilio.cc/index.php?n=Papilio.PapilioOne. (Accessed on 06/08/2017).

[20] "Papilio learning." http://papilio.cc/index.php?n=Papilio.Learning. (Accessed on 06/08/2017).

[21] P. J. Ashenden, *The Student's Guide to VHDL*. Elsevier/Morgan Kaufmann, 2 ed., 2008.